BIOINFORMATICS ORIGINAL PAPER Vol. 27 no. 20 2011, pages 2797–2805
doi:10.1093/bioinformatics/btr483

Sequence analysis Advance Access publication August 19, 2011

PTPan—overcoming memory limitations in oligonucleotide string matching for primer/probe design

Tilo Eißler, Christopher P. Hodges and Harald Meier*

Chair of Computer Architecture, Department of Informatics, Technische Universität München, Boltzmannstrasse 3, 85748 Garching, Germany

Associate Editor: John Quackenbush

ABSTRACT

Motivation: Nucleic acid diagnostics has high demands for non-heuristic exact and approximate oligonucleotide string matching concerning *in silico* primer/probe design in huge nucleic acid sequence collections. Unfortunately, public sequence repositories grow much faster than computer hardware performance and main memory capacity do. This growth imposes severe problems on existing oligonucleotide primer/probe design applications necessitating new approaches based on space-efficient indexing structures.

Results: We developed PTPan (spoken *Peter Pan*,'PT' is for *Position Tree*, the earlier name of suffix trees), a space-efficient indexing structure for approximate oligonucleotide string matching in nucleic acid sequence data. Based on suffix trees, it combines partitioning, truncation and a new suffix tree stream compression to deal with large amounts of aligned and unaligned data. PTPan operates efficiently in main memory and on secondary storage, balancing between memory consumption and runtime during construction and application. Based on PTPan, applications supporting similarity search and primer/probe design have been implemented, namely FindFamily, ProbeMatch and ProbeDesign. All three use a weighted Levenshtein distance metric for approximative queries to find and rate matches with indels as well as substitutions.

We integrated PTPan in the worldwide used software package ARB to demonstrate usability and performance. Comparing PTPan and the original ARB index for the very large ssu-rRNA database SILVA, we recognized a shorter construction time, extended functionality and dramatically reduced memory requirements at the price of expanded, but very reasonable query times.

PTPan enables indexing of huge nucleic acid sequence collections at reasonable application response times. Not being limited by main memory, PTPan constitutes a major advancement regarding rapid oligonucleotide string matching in primer/probe design now and in the future facing the enormous growth of molecular sequence data.

Availability: Supplementary Material, PTPan stand-alone library and ARB-PTPan binary on http://ptpan.lrr.in.tum.de/.

Contact: meierh@in.tum.de

Supplementary Information: Supplementary data are available at *Bioinformatics* online.

Received on May 11, 2011; revised on August 10, 2011; accepted on August 11, 2011

*To whom correspondence should be addressed.

1 INTRODUCTION

Fast methods for non-heuristic exact and approximate oligonucleotide string matching are of central importance for *in silico* primer/probe design and evaluation. Many applications in the field of nucleic acid diagnostics are based on the usage of indexing structures to speed up computations (Kurtz *et al.*, 2004; Ludwig *et al.*, 2004; Phillippy *et al.*, 2007). A prominent example is the PT-Server, the central search index of the software package ARB, which is used worldwide for phylogenetic microbial sequence analyses of molecular markers such as ribosomal RNA (Ludwig *et al.*, 2004). Based on suffix trees, the PT-Server is suitable for highly efficient approximate oligonucleotide string matching supporting the design and *in silico* evaluation of phylogenetic primers and probes widely applied in microbiology (Amaral-Zettler *et al.*, 2009; Kim *et al.*, 2010; Schönmann *et al.*, 2009). Furthermore, it is the basic component of the probeCheck server and the recently published comprehensive signature search tool CaSSiS (Bader *et al.*, 2011; Loy *et al.*, 2008).

The PT-Server has to fit into main memory during construction and application. This imposes limitations on the rapidly growing databases. Current collections of small subunit ribosomal RNA (ssu- rRNA) such as Greengenes (DeSantis *et al.*, 2006) or SILVA (Pruesse *et al.*, 2007) are already becoming unsuitable for common desktop computers. The growth of SILVA (See statistics at http://www.arb-silva.de/documentation/background/release-104/) reveals the fatality of the memory dependence in near future.

In addition, the PT-Server relies on Hamming distance metric for approximate string matching concerning primer/probe design and evaluation. The resulting inability to identify insertions or deletions (indels) prohibits the detection of possible sequencing errors or mutations which can lead to a misjudgment of the sensitivity of primers or probes with severe implications to experimental results (McIlroy *et al.*, 2011). Finally, the PT-Server can only be constructed from an ARB database limiting its field of application strictly to the ARB environment.

Consequently, a new memory efficient and sustainable approach is required, coping with the increasing sequence data flood. Extended Levenshtein (edit) distance metric search includes significant matches with indels and thus improves the future *in silico* prediction of sensitivity and specificity of an oligo primer/probe sequence. For the same reason, the new approach should support indexing of ambiguous sequence characters and treating matches at ambiguous positions in an efficient way. It seems reasonable to treat all ambiguities as wildcard in string matching ('N'-character).

Furthermore, features for advanced primer/probe design should be supported as well. These comprise the handling of alignment data. Beyond the precise comparative positioning of matches, it provides the opportunity to include higher order structure and function information, such as probe accessibility, in the *in silico* design process (Kumar *et al.*, 2005). Furthermore, a weight scheme for approximate matches is important for primer/probe design. Weighting approximate matches with respect to type and position of mismatches can help to scale the distance between exact and inexact matches as well as to better rate the specificity and sensitivity of a probe (Yilmaz *et al.*, 2008).

Many state-of-the-art indexing techniques, mainly originating from indexing structure theory, have been examined for their general viability to function as core structure for our approach taking into account aforementioned requirements.

The k-truncated suffix tree (*kTST*) (Schulz *et al.*, 2008) reduces the amount of required memory by limiting the depth of suffix trees like the PT-Server described above does, suffering from the same memory constraints. A data structure well known for being able to replace suffix trees is the enhanced suffix array (*eSA*). In practice, it has a dramatically reduced memory requirement of $6n$ bytes per input character (Abouelhoda *et al.*, 2004). Unfortunately, it still needs to fit into memory entirely during application. Taking the rRNA-sequence database SILVA (Pruesse *et al.*, 2007; release 104, SSURef_104_SILVA_08_10_10_opt.arb), with its 738 million nucleotide bases, this would result in a memory requirement of already >4 GB for the final index. Sophisticated approaches to further reduce main memory requirements are the so-called self-indexes, such as the compressed suffix trees (*CST*) (most recently Ohlebusch *et al.*, 2010). These use memory space in the order of input text size, already including the original text. Russo and colleagues, however, report a severe slowdown of factor 6.2 to 33 compared with 'classical indexes' conducting Levenshtein distance-based approximate string matching for error limits 1–3 on self-indexes, even for their fastest algorithm (Russo *et al.*, 2009).

Some approaches shift the memory requirements from main memory to secondary storage to build an index. The disk resident suffix arrays (*rSA*) with reduced space requirements offer a compact representation of suffix arrays on secondary storage (Moffat *et al.*, 2009). Moderate disk space requirements are reported for *rSA* coding DNA4. Exact pattern matching is functional but being more favorable for long query strings in terms of response time (Moffat *et al.*, 2009). However, no report on approximate string matching using *rSA* is published to date. Suffix trees in external memory (*eST*), recently reviewed by Barsky and colleagues (Barsky *et al.*, 2010), are well studied. *eST*s can be constructed efficiently even with limited main memory, although requirements for secondary storage are high. Approximate string matching, however, is reported to be an open challenge for existing *eST* approaches.

Searching for problems related to approximate oligonucleotide string matching, mapping short reads produced by next-generation sequencing technologies to reference sequences attracts attention. A representative mapping tool known to be among the fastest ones is *BWA* (Li and Durbin, 2009). It looks promising for oligo string matching in the field of primer/probe design as well. This is due to its fast query times, low main memory demands, its non-heuristic search capabilities and the ability to identify indels. *BWA* is currently incapable of indexing ambiguous characters or dealing with alignment information. However, an examination of its performance seems to be reasonable in order to decide on the viability for adding the missing features in a reimplementation.

In summary, with approximate oligonucleotide string matching as main purpose in mind, *eST* seems the most appealing approach while it still has remaining challenges, namely the memory consumption on secondary storage and the inability to perform approximate string matching reasonably. The first issue can be partly addressed by combining *eST* with truncation employed in *kTST* and the PT- Server. Nevertheless, approximate string matching still needs to be addressed. An initial approach addressing both challenges at once, conducted in our group, looked promising, but was limited to small databases (Hodges, 2003).

To keep up with the growth of available nucleic acid sequence data and to provide further search functionality enhancing the capabilities of existing oligonucleotide string matching approaches such as the ARB PT-Server, we developed PTPan, a space efficient search index capable of dealing with large amounts of nucleic acid (RNA/DNA) data. Based on suffix trees on secondary storage, it combines partitioning, truncation and a new suffix tree stream compression to construct a space-efficient index. It may outgrow available main memory while enabling efficient approximate oligonucleotide string matching, solving the issues of suffix trees on secondary storage for our field of application. We implemented the applications ProbeMatch and ProbeDesign as well as FindFamily based on PTPan, utilizing the Levenshtein distance metric for approximate oligonucleotide string matching. This allows to identify indels besides substituted base positions. All ambiguous base positions in the sequence are treated as wildcard ('N'). Results are formatted user-friendly for easy evaluation.

PTPan is implemented in C/C++. It is available as stand-alone library and incorporated into the ARB software environment where it can be used as a replacement for the PT-Server. It runs on 64-bit (Ubuntu) Linux systems which are common nowadays.

2 METHODS

2.1 Suffix tree basics

The suffix tree is a well-known index structure. For a non-empty string S over the alphabet Σ, a suffix tree is a tree with labeled edges. The labels contain one or more characters. The paths of the tree, i.e. the concatenation of edge labels from the root to a leaf, correspond to the suffixes of S. Their start positions are stored at the leaves. Therefore, a suffix tree allows for fast searching of substrings contained in S taking time linear to the length of the search string plus the number of occurrences in S to return all appearances. A recent review by Barsky *et al.* (2010) includes a more detailed description of suffix trees and common construction algorithms.

2.2 Sequence alphabets and compact representation

Nucleic acid sequences are concatenations of symbols out of the DNA4 alphabet consisting of the four symbols 'A', 'C', 'G' and 'T' (or 'U' in RNA, respectively). Many nucleic acid sequences in public repositories contain ambiguous bases which are all represented by 'N' within PTPan. The corresponding alphabet, DNA5, is used in this study.

For a compact representation, the codes of the DNA5 alphabet are first mapped into the following integer interpretation: val('N') = 0, val('A') = 1, val('C') = 2, val('G') = 3, val('T') = val('U') = 4. A series of bases can now be stored as the sum of their sequence codes to the base of five: $pval = \sum_{i=1}^{n} 5^{n-i} * val(S_i)$. This representation allows to store a sequence of 27 bases in a 64-bit integer value (as $5^{27} < 2^{64}$) leading to a usage

of $\frac{\log 5}{\log 2} \simeq 2.3$ bits per base compared with 1 byte per character in the default representation of strings.

2.3 Branch mask

To avoid the storing of empty downward pointers in a tree, a branch mask is used. This branch mask of alphabet size $|\Sigma|$ has the bit i set for each element i of the alphabet. For DNA5, this results in a mask of five bits. Hence, the first base character of every downward edge is already defined implicitly and does not need to be stored in the edge label.

2.4 Encodings

Prefix codes are applied to provide variable length payload information after the corresponding prefix. Prefixes must conform to the requirement demanding no prefix being prefix of another one.

Huffman encoding is used to represent the sequence data losslessly according to its entropy (Huffman, 1952). This technique is used e.g. to store more frequently occurring branch masks in the compressed stream with a shorter representation than less frequent ones reducing the number of bits required on average. Further Huffman encoding is utilized for the final compressed suffix tree representation (Section 3.3.2).

Delta encoding calculates the difference between one value and the next in an array instead of storing the full value. For example, a list of suffix occurrences inside a sequence 170 200, 170 385, 170 544, etc. is stored as 170 200, 185, 159, etc. in the compressed suffix tree.

3 RESULTS

PTPan combines several techniques to construct a persistent and space-efficient suffix tree. The core structure is a partitioned, truncated and stream-compressed suffix tree. Utilizing the suffix tree, approximate string matching algorithms as well as primer/probe design functionality have been implemented.

The three main steps of the PTPan construction algorithm are as follows:

(1) Data retrieval and preparation (Section 3.1).
(2) Partition determination and index header storage (Section 3.2).
(3) Constructing compressed suffix tree for each partition (Section 3.3).

3.1 Data retrieval and preparation

PTPan incorporates distinct sequences into one index. The sequences are retrieved by a defined interface currently available for the ARB database and for multiFASTA-files. A compact representation of the data is stored in the index header utilizing the compression method described in Section 2.2. Optionally available alignment information is included as well, i.e. gap characters and a reference entry.

For the construction algorithm, all sequence data are merged into a temporary compressed raw data file stripping all non-DNA5 characters. For referencing the position inside the merged raw sequence data to its origin, the start position is stored for each sequence entry. The raw data is padded with 'N'-characters at the end to ease the construction process (Section 3.3.1).

3.2 Partition determination and index header storage

First, the maximum number of base positions fitting into available main memory is calculated. If the merged raw sequence (Section 3.1) has more bases than this threshold value, partitioning is required. If so, the merged raw sequence (Section 3.1) is scanned once for the frequency of each prefix of a fixed, data-dependent precalculated length. This histogram information is then used to determine partitions small enough to fit into main memory during construction. Prefixes start with length one, i.e. one for each symbol of the alphabet. These initial prefixes are extended by one character for each symbol of the alphabet if the number of occurrences exceeds the threshold value. For all newly created prefixes, the determination process is repeated until all defined partitions fit into main memory. Prefixes not occurring in the sequence data are omitted.

Finally, all remaining basic index information is stored into the index header including the number of partitions and their prefixes.

3.3 Constructing compressed suffix tree for each partition

For each calculated partition, a truncated suffix tree is constructed in main memory. Thereby, two kinds of suffix nodes are distinguished (Section 2.1). Inner nodes have up to five children, but no leaves. A branch mask (Section 2.3) keeps track of the downward children present. Border nodes have no children, but at least one leaf. Both node types carry ingoing edge information, i.e. label and length, except the root node. The label is stored as 64-bit integer value during construction instead of a reference to the original source text. This allows an edge-label length of up to 27 characters (Section 2.2).

The default pruning depth of the suffix tree is set to a path length of 20. This value can be altered, though the current implementation is limited to a maximum length of 27 for a path.

3.3.1 Insert prefix occurrence For each partition, defined by its prefix, the input sequence is scanned once. A sliding window is used to avoid repeated accesses to the raw sequence data. It keeps a copy of enough parts of the compressed raw sequence data to obtain the sequence string starting with the current position and the following characters up to the pruning length. If an occurrence of the current prefix is reached, the existing suffix tree is traversed comparing the edge labels with the sequence starting at the position. If a border node is reached, the path is already present and the current occurrence is added as new leaf. If a mismatch appeared inside an inner node, a new border node including one leaf is added with the remaining characters of the window as edge label. The last case is a mismatch inside an edge label. It requires splitting of the edge label, insertion of a new inner node labeled with the matching part, update of the downstream node and insertion of a new border node for the remaining characters of the current window.

To avoid the repeated traversal of the first few levels of the suffix tree, a hash-map is used for the suffix–prefixes up to a certain length available as integer values in the compact representation (Section 2.2). The hash entry for a prefix points to the appropriate inner node which is reached by traversing from the root down the path denoted by the characters of the prefix. If a prefix of a new suffix to insert is already present, several edge comparisons can be omitted. The insertion process can start to add the rest of the suffix after the prefix at the appropriate inner node. This speeds up the construction algorithm significantly.

3.3.2 Prepare edges and branch masks For the final stream-compressed suffix tree, two kind of edges are distinguished. *Short edges* are all edges with less or equal length of six characters.

Instead of storing an offset to a dictionary string and the length of the edge, a statistically based Huffman encoding of the edge is stored directly into the compressed stream as it is more space efficient. This takes into account that occurrences of short edges are not equally distributed. Scarce short edges are not encoded, keeping the Huffman tree representation compact. *Long edges* are all edges that do not get a short edge representation, either because they are too long to store directly or they do not appear frequently enough. Long edges are stored as a combination of an offset inside a dictionary string and the length of the edge. A long edge dictionary string is stored along with the suffix tree of the corresponding partition.

Before writing the tree to disk, long and short edges as well as the branch masks are prepared. The in-memory suffix tree is traversed counting the short edges and the occurrences of the different branch mask combinations. This enables the calculation of the Huffman encoding (Section 2.4) for the short edges, providing a single one for long edges as well, and for the branch masks. Afterwards, during a second traversal, the long edges are counted and the short edges are replaced by their Huffman representation.

3.3.3 Building the long edge dictionary PTPan utilizes a greedy algorithm for building the long edge dictionary. Initially, the edge labels are sorted lexicographically enabling the locating of edges that are equal or prefixes to other ones. These prefix edges are clustered together by equating the offsets. Afterwards, the edges are sorted by starting offset. This allows the detection of overlapping fragments. These fragments are chained to longer concatenated edge labels for the dictionary, covering the smaller edge fragments.

After building the concatenated edges, these are sorted by length. An initial subset of them is added to the dictionary and a temporary truncated suffix tree is constructed allowing fast determination if an edge is already present in the dictionary. Afterwards, the remainder of the concatenated edges are looked up and added to the long edge dictionary if not already present. The temporary suffix tree is updated every time a new edge has been added.

Subsequently, the long edges are sorted by their length ensuring that as many edges as possible are a substring of prior inserted ones. Long edges which are part of a concatenated edge are skipped in this step as they are already present in the long edge dictionary. For the remaining ones, the temporary suffix tree is queried for an occurrence. If none is found, the edge is appended to the dictionary.

Finally, the long edge dictionary is compressed (Section 2.2) and a long edge length Huffman encoding is built, which is required for storing.

3.3.4 Relocating the tree In the final suffix tree representation, the downward child pointers are stored as offsets relative to the position right after the current node. Thus, the size of all child nodes must be available prior to storing a node. Therefore, the tree is traversed in depth-first-search (DFS) order as final preparation step prior to writing the suffix tree. During this traversal, the size of each node is calculated and stored instead of the downward child pointer of the parent node. The tree DFS order is kept in an array of pointers to the suffix nodes for writing the tree to disk.

3.3.5 Writing the stream-compressed suffix tree The suffix tree is written to disk as a stream of bits. During the compression step, the suffix nodes used during construction are transformed into compressed nodes. A compressed node consists of three parts: the edge label, the branch mask and the child references, for inner nodes respectively, a compressed leaf array for border nodes.

For nodes with a short ingoing edge, its corresponding Huffman encoding is written into the stream. Otherwise the stream starts with the special Huffman code for long edges followed by the Huffman encoded edge length and the long edge dictionary offset.

Afterwards, for inner nodes, the branch mask Huffman code is issued then followed by up to five encoded child reference offsets in DFS order corresponding to the branch mask bits. As the references are written in ascending order, each can be stored relative to the last one reducing space requirements. In case of a border node, a leaf array follows instead of the child references indicated by a special code for the branch mask. To store the leaf array in a more efficient way, the sequence offsets are sorted in ascending order. Afterwards, the resulting array is delta encoded (Section 2.4) to obtain smaller values for storing. A detailed example of the compression can be found in the Supplementary Figure S1.

3.4 PTPan applications

3.4.1 Basic tree traversal For each partition, the root node is decompressed as it starts at position 0. All child node offsets are revealed during this and the tree can therefore be traversed recursively top–down to the nodes carrying the leaf arrays. The decompression of the stream is the inverse operation to the encoding described in Section 3.3.5. Because of the branch mask, the first character of a downward edge can be determined prior to decompressing the next node, one level deeper into the tree.

3.4.2 ProbeMatch is a string matching algorithm leveraging basic tree traversal described in Section 3.4.1. It is capable of carrying out exact as well as approximate string matching using the Levenshtein distance metric. This enables the locating of insertions and deletions (indels) besides substitutions.

Approximate string matching can be executed in basic- or weighted-match mode. In basic-match mode, each substitution or indel increase mismatch counter by one. In weighted-match mode, the type of a substitution and its position impacts the size of a weighted mismatch score. In this mode, a fixed score value is assigned to indels currently. In addition, ‘N’s occurring in the original sequence are treated as wildcard and named ‘N’-mismatches which are counted separately.

String matching is carried out by searching the partitions one after another and merging the results into one list. Hits crossing entry borders in the merged raw sequence data are filtered out. Searching for query strings longer than the pruning depth requires a verification step for each hit. The query string is compared with the decompressed original sequence data of each hit candidate.

Subsequently, a result list representation with individual information on every match is created. It provides identifier and name of the sequence entry containing the match, a differential alignment representing equal positions, substituted bases, the position of a deletion (‘_’) or insertion (‘∗’) and match context at nine positions at $3'$ and $5'$ ends. The match position is returned as well, taking into account alignment information if available. Additionally, values indicating the distance to the search string are determined for each match. The first value ‘mis’ indicates the distance between search string and match by the total number of substitutions, insertions and deletions. The second value ‘wmis’

indicates the weighted distance according to the weighted mismatch score mentioned above. Both values are used for sorting the result list. Finally, 'nmis' displays the number of 'N'-mismatches.

3.4.3 ProbeDesign was implemented utilizing the PTPan index structure and ProbeMatch functionality. It allows to identify unique oligonucleotide probe sequences and their reverse complement, i.e. the target sequences. Goal is the identification of probe sequences with high coverage, i.e. preferably hitting a user selected single sequence or sequence group entirely, while guaranteeing high specificity, i.e. preferably hitting no out-group sequence or only with a high distance. The search can be initially constrained by setting parameters melting temperature, GC content and probe length.

An initial set of probe candidates matching these constraints is selected utilizing basic tree traversal (Section 3.4.1). The candidates are afterwards evaluated according to the settings of two additional parameters, i.e. minimum percentage of in-group and maximum number of out-group hits. Probe candidates not matching the latter constraints are omitted at this stage. Remaining candidates are matched with ProbeMatch functionality (Section 3.4.2) with up to four mismatches allowed. From the match lists, the number of out-group hits with 'wmis'-values ranging from 0 to 4 are counted and displayed at 'wmis'-value intervals of 0.2. Finally, all candidates are evaluated based on the percentage of in-group hits, the number of out-group hits and the parameters mentioned before. Afterwards a sorted list is displayed with the most promising probe candidates on top. The candidate list includes target and probe sequences, hit information and a list of out-group hit numbers for increasing weighted mismatch values.

3.4.4 FindFamily is a method for fast and comprehensive sequence similarity searches by oligonucleotide string matching frequencies. It can be used for unknown sequences to rapidly find their most similar sequences within large datasets without prior alignment. For a selected sequence entry, FindFamily generates oligonucleotide sequences of user-defined length utilizing a sliding window. Subsequently, depending on the settings, exact and approximate string matching is performed for each oligo, summing up the number of hits individually in a hit score for every sequence entry incorporated in PTPan. Finally, a result list returns the sequence entries sorted by decreasing hit score.

4 EXPERIMENTAL RESULTS

4.1 Experimental setup

PTPan was integrated into the ARB software package (Ludwig *et al.*, 2004) to compete with the current PT-Server. We compared time consumption and space requirements of the construction algorithms as well as oligonucleotide string matching times for both.

Real life test data were retrieved from the SSURef_104_SILVA database (Pruesse *et al.*, 2007, available at http://www.arb-silva.de/download/arb-files/). It contains >512 000 ssu-rRNA sequence entries with 1426 base characters average length, >738 millions in total. We created database subsets ranging from 1000 sequence entries to the complete SILVA database.

The main performance tests were executed on an Intel Core i7 920 with 2.67 GHz computer with 24 GB of main memory running 64-bit Ubuntu 10.04 LTS. This machine has been chosen to analyze the performance of PTPan and PT-Server without experiencing biases resulting from main memory limitations. Further spot tests have been carried out on an Intel Core 2 Duo T9400 with 2.53 GHz and 4 GB of main memory for measuring the performance taking influences from memory limitations into account, particularly expected for the full SSURef_104_SILVA.

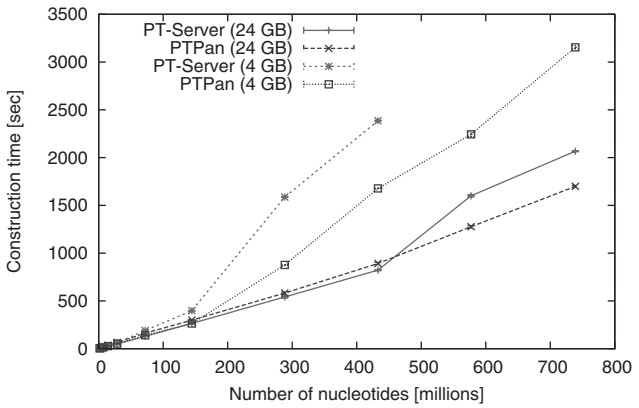

Fig. 1. Construction runtime with SD of constructing PTPan and PT-Server depending on dataset size in numbers of base characters for two different test computers. Datasets are based on SSURef_104_SILVA.

4.2 Construction time

The construction performance was measured for 10 replicates of PTPan and the PT-Server for each of the different test databases. Figure 1 plots the average value of the time to construct with standard deviation against the test dataset size in a number of nucleotides. An almost equal construction time was recognized up to dataset with 460 million base characters (equivalent with 300 000 sequence entries). For the two largest datasets tested, PTPan was constructed significantly faster than the PT-Server. This effect can be put down to a strong rise of the PT-Server construction time above 468 million nucleotides, caused by a construction algorithm alteration circumventing more disadvantageous memory effects (R. Westram, personal communication).

On the test computer with 4 GB main memory, PTPan was constructed for all test datasets, even the complete SSURef_104_SILVA database in 52 min. The construction time was significantly longer than on the 24 GB-memory machine, because a higher number of smaller partition had to be constructed, due to the limited memory size. The PT-Server, however, was efficiently constructed for smaller sized datasets only. For test datasets with 300 000 sequence entries or more, the PT-Server construction process started to use swap memory and could not finish in reasonable time starting with the 400 000 sequence entry database (Section 3.1 in Supplementary Material).

Further sustainability tests for the larger SSUParc_104_SILVA database with over 1.47 million entries and >1.45 billion nucleotides reveal an increase of construction time to about four h on the 4 GB test computer and 1 h on the 24 GB computer (details in Section 3.1 Supplementary Material).

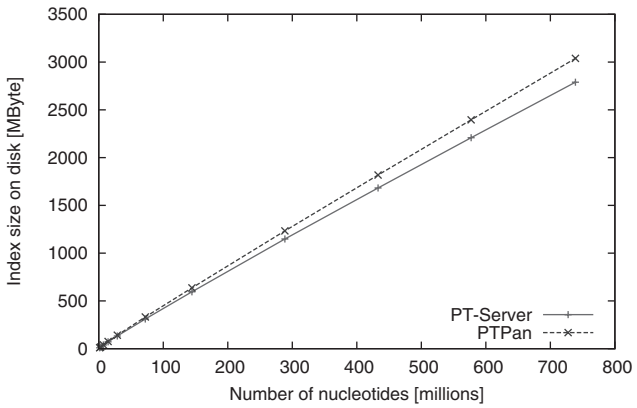

Fig. 2. Secondary storage requirements of PTPan and PT-Server.

4.3 Space requirements

In Figure 2, the space requirements on secondary storage are shown for the PTPan and the PT-Server. For the complete SSURef_104_SILVA database, PTPan requires about ∼3 GB, whereas the PT-Server requires ∼2.8 GB. The average memory consumption over all test databases is 5.1 bytes per base for PTPan and 4.9 bytes for the PT-Server.

The PT-Server is stored compressed. Before application, it is loaded and decompressed into main memory entirely in addition to the corresponding ARB database which must be present at the same time. For the usage of the complete SSURef_104_SILVA database including PT-Server, 5 GB out of 8 GB total memory use are exclusively for the PT-Server.

PTPan, in contrast, does not need to fit entirely into main memory during application due to its partitioned and stream-compressed design. The partition files are mapped into main memory to keep the memory demands low. But even when loaded entirely into main memory, PTPan required significantly less memory compared with the PT-Server. For the complete SSURef_104_SILVA it allocated 3 GB only, about the same memory space requirement as on disk.

Sustainability tests conducted with SSUParc_104_SILVA database reveal a doubling of the PTPan index size, thus increasing linear (details in Section 3.1 in Supplementary Material).

4.4 ProbeDesign

ProbeDesign offers fast search capabilities for unique oligonucleotide probes. It aims at finding probes matching most target sequence entries selected combined with a large distance to non-target entries, satisfying relaxed search parameters. The probe candidate result list is sorted according to high target group coverage and a large weighted distance to non-target sequence entries pinpointing the most promising candidates at the top positions (Fig. 3).

We examined performance of ProbeDesign using PTPan indexed SSURef_104_SILVA for 'best' and 'worst' case queries.

Searching for a unique 18mer probe sequence for a single sequence entry (accession number *AJ318041*) with default settings served as 'best' case query. ProbeDesign offered several probe candidates matching the search constraints and returned them in a sorted list with the probe sequence *CGGAGAGCAUAACGCCCU* on top (Section 3.3 in Supplementary Material).

As a 'worst' test case for probe design, the phylogenetic group containing all 16S rRNA sequences of domain 'bacteria' incorporated in the SSURef_104_SILVA database (almost 440 000 entries) has been chosen. The number of allowed out-group hits was set to 10, the coverage percentage to 75%. For the remaining values, default settings were kept. Figure 3 lists the probe candidates calculated in only about 5 min. The second suggested probe is the already published EUB338 targeting bacteria (Amann and Fuchs, 2008). This demonstrates that ProbeDesign is not only capable of fast generating probe candidates, but also of finding highly used probes confirming the correctness of the results.

4.5 ProbeMatch

String matching capabilities of PTPan and PT-Server were tested with the SSURef_104_SILVA database for proposed 'best' case and 'worst' case scenarios. Here, the single-sequence-specific probe designed in Section 4.4 served as 'best' case for its property to have minimal number of hits in the test database. The 'worst' case example was the probe EUB338, a sequence well known to have a very high number of entry hits in SILVA (Amann and Fuchs, 2008). Tests have been conducted in basic-match mode (section 3.4.2) for exact as well as approximate queries up to five allowed mismatches.

In Figure 4, quantitative aspects of ProbeMatch results are shown. The hit numbers are plotted against the numbers of non-weighted mismatches defined as upper limit for a hit. For exact queries, the number of hits are the same. For approximate string matching, PTPan returns a higher number of hits with the divergence increasing for higher number of mismatches allowed. The discrepancy is caused by the PT-Server utilizing the Hamming distance metric while PTPan leverages the Levenshtein distance metric. Comparing PT-Server to PTPan, the latter additionally identifies matches with indels (details in Section 3.2 in Supplementary Material).

Figure 5 shows overall string matching times for different numbers of maximum allowed mismatches. This includes the time required to generate the result list. PTPan shows about four times higher string matching times for exact queries. For approximate string matching with error rates from 1 to 4, PTPan is 8.5–15 times slower compared to the PT-Server extending to about 32–33 times for a maximum error rate of five. One reason is the requirement of PTPan decompression during traversal. Furthermore, for higher error rates, the number of paths in the suffix tree containing possible hits increases faster as PTPan must check for indels besides substitutions. Finally, for the larger amount of hits, output creation takes longer along with differential alignment creation being more complex. Hit context information needs to be decompressed and, if mismatches are allowed, positions of indels must be determined. Compared to locating substituted bases, this is more compute intensive. Figure 6 shows an excerpt of a PTPan result list incorporating indel hits.

Sustainability tests were conducted on SSUParc_104_SILVA database revealing constant matching times per hit returned compared with SSURef_104_SILVA (Section 3.2 in Supplementary Material).

Evaluating *BWA* revealed faster matching times for exact and approximate string matching with default settings compared with PTPan. Though *BWA* returns only the single query hit with the lowest distance to the search string. Modifying *BWA* settings to be comparable with PTPan basic-match mode, i.e. disabling iterative

```
Length of probe      18
Temperature          [30.0 -100.0 ]
GC-Content           [50.0%-100.0%]
E.Coli Position      [any]
Max Non Group Hits     10
Min Group Hits         75%
Target              le      apos ecol   grps  G+C 4GC+2AT Probe sequence     | # of non-group hits for increasing ProbeMatch weighted mismatch values
AUGUUCGGUUAAGUCCCG 18 A= 34462 1082 328363 50.0 54.0    CGGGACUUAACCCAACAU |  0;  0;  0;  0;  0;  0;  0;  0;  0;  1;  6;  6; 27; 78;114;641;1479;2914;3801;5230;
ACUCCUACGGGAGGCAGC 18 B=  6334  338 398806 66.7 60.0    GCUGCCUCCCGUAGGAGU |  0;  0;  2; 19; 20; 20; 28; 38; 62; 75;154;726;1813;1915;2385;3807;16706;17649;1943
CGGUGAAUACGUUCCCGG 18 C= 41541 1369 328016 61.1 58.0    CCGGGAACGUAUUCACCG |  0;  1;  1; 25; 25; 26; 33;306;314;327;7910;8061;8103;8347;9844;11515;13023;13615;1
CUCCUACGGGAGGCAGCA 18 D=  6386  339 398874 66.7 60.0    UGCUGCCUCCCGUAGGAG |  0;  2; 14; 22; 28; 37; 38; 63; 72;111;326;1134;1885;2325;15715;16300;17070;18849;2
CUCAGCUCGUGUCGUGAC 18 E= 34124 1064 329352 61.1 58.0    CUCACGACACGAGCUGAC |  0;  0; 35; 99;113;120;426;427;865;3256;3306;3678;4674;4717;4956;8527;8946;9545;122
CUCGUCAGCUCGUGUCGU 18 E-     9 1061 329090 61.1 58.0    ACGACACGAGCUGACGAC |  0;  3; 15; 25;357;391;466;1110;1304;3808;3877;4289;5141;6551;11159;12567;22432;238
UCGUCAGCUCGUGUCGUG 18 E-     4 1062 328962 61.1 58.0    CACGACACGAGCUGACGA |  0;  4; 67; 91;111;344;776;894;1078;3468;3531;3737;4252;5432;5501;9533;11904;13780;
CGUCAGCUCGUGUCGUGA 18 E-     3 1063 328773 61.1 58.0    UCACGACACGAGCUGACG |  0;  0; 66;116;118;430;775;862;959;3490;3621;3649;3900;4958;8274;10030;10436;10771;
AGCAGCCGCCGUAAUACG 18 F= 13130  520 358200 61.1 58.0    CGUAUUACGCCGGCUGCU |  0;726;743;753;789;48321;59630;59749;59792;60896;62563;71090;71411;72756;72879;7354
ACAUACCCUGCUACUCCA 18 G= 23959  790 341474 50.0 54.0    UGGACUACCAGGGUAUCU |  0; 29; 30;1242;1249;16909;17515;17640;19100;19489;19629;19741;20161;20696;21565;21
UCAGCUCGUGUCGUGAGA 18 E+     5 1065 328783 55.6 56.0    UCUCACGACACGAGCUGA |  0; 26; 37; 37;101;104;421;449;3026;3280;3305;4302;4374;4761;4907;5489;8908;12175;1
UCCUACGGGAGGCAGCAG 18 D+     2  340 398409 66.7 60.0    CUGCUGCCUCCCGUAGGA |  6; 26; 42; 42; 43; 44; 84;100;138;306;1131;14211;15546;15830;16122;19566;21226;223
CCUACGGGAGGCAGCAGU 18 D+     3  341 392674 66.7 60.0    ACUGCUGCCUCCCGUAGG | 23; 64; 64; 65; 66; 86;152;165;200;238;3380;14745;15673;15853;19312;28335;28819;299
CUCAAAGGAAUUGACGGG 18 H= 28426  910 339444 50.0 54.0    CCCGUCAAUUCCUUUGAG | 23;451;19501;61862;63040;63260;63454;63803;63946;64199;64582;64939;65426;74163;7545
UCAAAGGAAUUGACGCGG 18 H+     4  911 334290 50.0 54.0    CCCCGUCAAUUCCUUUGA | 24;308;19094;19654;20272;20335;20534;63387;63743;64052;64438;65188;65419;65762;6617
```

Fig. 3. ProbeDesign: screenshot of oligonucleotide probe and target sequence candidates for domain 'bacteria' in SSURef_104_SILVA database. Sensitivity can be evaluated with the information about in-group coverage ('grps'), localization in respect to a reference (alignment) entry ('apos' and 'ecol'), melting temperature ('4GC + 2AT') and GC content ('G + C'). For evaluating the specificity of each target/probe sequence candidate, the numbers of non-group hits are computed for increasing 'wmis'-values (see Section 3.4.3), and displayed on the right-hand side column of Fig. 3. Increasing weighted mismatch values.

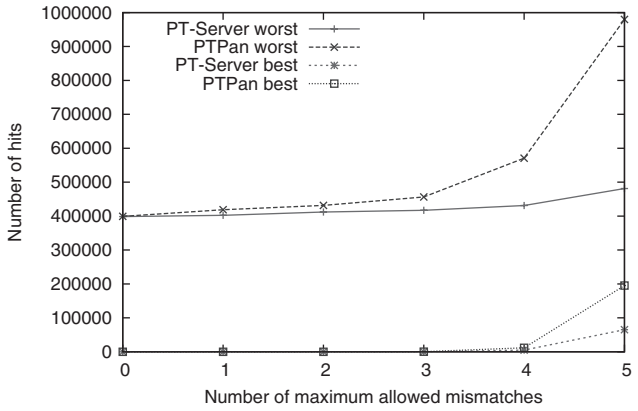

Fig. 4. Number of query hits for PTPan and PT-Server searching in basic-match mode the SSURef_104_SILVA database for EUB338 ('worst' case) and a one entry specific probe ('best' case).

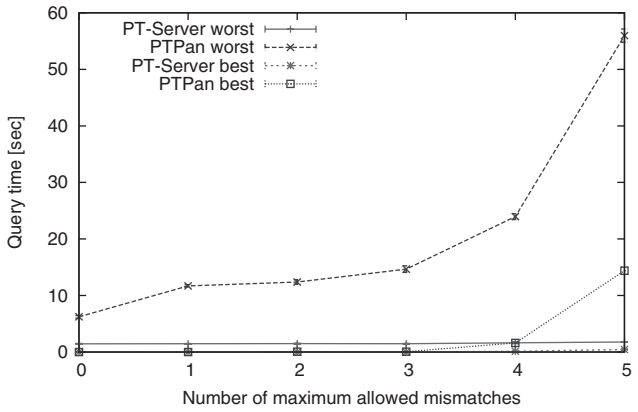

Fig. 5. Query times and SD for PTPan and PT-Server basic-match mode searching SSURef_104_SILVA database for EUB338 ('worst' case) and a unique example probe hitting only one entry exact ('best' case).

search in order to return all matches, lowering mismatch and gap penalties as well as allowing more than one gap open, resulted in approximate string matching times of *BWA* being worse compared with PTPan. Furthermore, the *BWA* result list does not include information about the weighted mismatch value, mismatch type and position, mismatch context at 3′ and 5′ ends as well as alignment-related position information. Details about *BWA* test results and settings are reported in Section 1 in Supplementary Material .

4.6 FindFamily

To test FindFamily, different sequence entries have been chosen in the SSURef_104_SILVA database (Pruesse *et al.*, 2007). For each chosen entry, FindFamily has been carried out and the results were evaluated by comparing the proposed next neighbors to the neighbors in the phylogenetic reference tree which is incorporated into the SILVA database. This process was repeated for > 50 entries. The average runtime was <1 min with default settings. The result listed, in all cases, the next neighbors in the phylogenetic tree as candidates. Thus, the results show that FindFamily delivers a good first estimation of sequence similarity (details in Section 3.4 in Supplementary Material).

5 DISCUSSION

We presented PTPan, a persistent and space-efficient suffix index for oligonucleotide string matching which deals with limited main memory. It incorporates enhanced non-heuristic exact and approximate search capabilities to find exact as well as approximate matches containing substitutions and indels, due to a user-defined search string distance. It has been developed and optimized to support similarity searches and primer/probe design in huge nucleic acid sequence collections, highly demanded in molecular microbial diagnostics. PTPan is integrated into the ARB software package (Ludwig *et al.*, 2004) and available as stand-alone library.

We have reviewed memory-efficient indexing structures according to their viability for non-heuristic approximate oligonucleotide string matching with respect to our main focus, primer/probe design in large nucleic acid sequence databases (Section 1). In doing so, we have evaluated *BWA* for suitability

```
ButFibri Butyrivibrio fibrisolvens          1   0  1.6  6334  338 0  ACGGCCCAA-=====*==========-AGUGGCGAA
NonDietz Nonomuraea dietziae                1   0  1.6  6334  338 0  ACGGCCCAG-===_==========-AGUGGCGAA
SpcBrevi Sporichthya brevicatena            1   0  1.6  6334  338 0  ACGGCCCAG-===_==========-AGUGGCGAA
AtmFormo Actinomadura formosensis           1   0  1.6  6334  338 0  ACGGCCCAG-===_==========-AGNGGCGAA
UnlBa819 uncultured bacterium               1   0  1.6  6334  338 0  ACGGCCCAG-=========*========-AGUGGCGAA
EntSpeci Enterococcus sp.                   1   0  1.6  6334  338 0  ACGGCCCAA-==========*=======-AGUAGCGAU
```

Fig. 6. ProbeMatch: excerpt from PTPan generated result list searching the SSURef_104_SILVA database for the EUB338 target sequence with one allowed mismatch in basic-match mode. The result list shows matches with insertions ('∗') or deletions ('_').

for our approach in practice. *BWA* is a successful non-heuristic short read mapper based on a space-saving index structure with fast construction times (Li and Durbin, 2009). Although short read mapping is a problem related to oligonucleotide string matching for primer/probe evaluation, it differs in the actual goal. *BWA* aims at the best hit for a short read. In contrast, primer/probe design for microbial diagnostics targets at finding all matches with distance smaller or equal to a required value to the query string. Every match is important to detect *in silico* sensitivity and specificity of a primer/probe as good as possible, in particular to identify potentially cross-reacting non-target sequences (and respective organisms) in the analyzed data collection.

BWA performed approximate string matching faster than PTPan for default settings, though missing significant matches. To get thorough enough results for careful primer/probe evaluation, *BWA* parameter settings had to be adopted resulting in a performance worse than PTPan. Furthermore, essential information for advanced evaluation of primer/probe candidates is not provided (see Section 4.4). Extending *BWA* by required functions would lead to a further incalculable slowdown, as we concluded.

We focused our further efforts on suffix trees on secondary storage. The remaining challenges of high space requirements and inability to efficiently carry out approximate string matching were targeted. We employ partitioning and truncation combined with a new suffix-tree stream compression to reduce space requirements and optimizing the index structure for top–down traversal required by approximate oligonucleotide string matching. Though the achieved memory efficiency has an implication for the current implementation of ProbeDesign, the length of probes is limited to the depth of the suffix tree, 27 at maximum. Nevertheless, this is sufficient for many applications such as PCR or fluorescence *in situ* hybridization (FISH) (Amann and Fuchs, 2008). Designing longer probes is possible using a sideway if no other options exist. ProbeMatch results, e.g. of a 27mer probe target candidate, provide context information of each match. With this a target candidate sequence can be elongated according to the matching base characters within the context. Since ProbeMatch is not limited to 27mers, this procedure can be repeated iteratively in order to design longer oligonucleotide probes. Alternatively, concatenation of overlapping 27mer probe target sequence candidates could be applied.

An index providing comparable functionality targeting the same field of application, namely primer/probe design based on oligonucleotide approximate string matching, is the widely used PT-Server integrated into the ARB software package (Ludwig *et al.*, 2004). A detailed comparison between PT-Server and PTPan revealed similarities as well as important differences.

As public nucleic acid sequences may contain ambiguous base characters, proper handling is required. Both indexes treat all ambiguities as 'N'-character during build and wild-card during search. The great advantage of this is the identification of matches containing ambiguous characters which helps to optimize *in silico* primer/probe evaluation.

In addition, both indexes support basic- and weighted-match mode (Section 3.4.2). This is extremely advantageous during search and evaluation of diagnostic oligonucleotide primer/probe candidate sequences. In particular, a weighted mismatch allows to measure a fine grain distance between a match and the search sequence string, taking type and position of substitutions into account. This enables rating of *in silico* specificity and sensitivity of a probe by evaluating its distance to target and non-target sequences. This is important when predicting the ability of an oligonucleotide probe to cross-react with a non-target oligonucleotide sequence stretch despite mismatches (Yilmaz *et al.*, 2008).

Both PTPan and the PT-Server return a meaningful result list for each query, facilitating fast evaluation of query hits significantly. The list incorporates hit names, absolute and relative position in the original multiple alignment, number of mismatches, weighted mismatch scores and the sequence contexts in a multiple differential alignment. For list creation, the original sequence data are required to retrieve context information. To avoid memory intensive permanent accesses to the original database during application—as with the PT-Server—PTPan incorporates all sequence and alignment information making it independent of the sequence data source.

Besides the similarities, there are some major advantages of PTPan compared with the PT-Server. For approximate string matching, the PT-Server utilizes the Hamming distance metric for finding exact matches and those with substituted bases. However, it misses matches with indels. This could be a major drawback when trying to assess *in silico* coverage and specificity of diagnostic oligonucleotide primer/probes. McIlroy and colleagues have recently shown that oligo probes can form stable hybrids with non-target sequences despite indels, leading to false positives in FISH (McIlroy *et al.*, 2011). Thus, the ability to include indels into the probe design and evaluation process is of great interest. PTPan, utilizing the Levenshtein distance metric, also detects matches with real indels and those resulting from sequencing errors at the target position (Sections 3.4.2 and 4.5). This is a major advancement for *in silico* primer/probe design.

The PT-Server relies entirely on source sequences in the ARB database format for index construction. PTPan currently supports ARB databases and multiFASTA files. Moreover, it provides an interface easy to implement for other sequence data formats. PTPan is available as stand-alone library, too. Hence, it is easily usable for oligo string matching beyond the ARB software environment.

Finally, PTPans central advantage over the PT-Server is its significantly lower main memory demand during index construction and application (Section 4.3). The PT-Server has comparable memory requirements on disk as it reduces space consumption by truncation, but significantly higher demands in main memory (Section 4.3). This causes severe problems when trying to apply

the PT-Server to approximate oligonucleotide string matching on real-life datasets on a common desktop computer, which is solved by PTPan (Section 4.2). In the case of the thousands of ARB users worldwide, an index for the complete SSURef_104_SILVA database with >512 000 sequence entries and 738 million nucleotides can now be constructed and used for string matching, probe design and sequence similarity searches, even with only 4 GB main memory instead of at least 8 GB required before (Sections 4.2 and 4.3). PTPan clearly relaxes the problem of providing efficient, main memory-independent string matching capabilities facing the nucleic acid sequence data growing faster than computer hardware performance and main memory capacity.

Regarding construction time, PTPan is highly competitive to the PT-Server despite the additional load of partitioning and compression (see Section 4.2). As expected, query times are slower by factor 4–33 depending on the number of mismatches allowed to define a match (Section 4.5). This is due to overhead induced by decompression and computationally intensive indel searches. In addition, PTPan identifies and returns a significantly higher number of hits for approximate string matching, i.e. indel matches besides matches delivered by the PT-Server. Regarding the consequences of query times slowdown for *in silico* probe design, it is important that usually the candidate sequences have to be evaluated down to matches with a maximum weighted distance of four (Yilmaz *et al.*, 2008). Thus, oligonucleotide string matching with five non-weighted mismatches, currently showing unacceptable performance, is rarely applied. Furthermore, PTPan query times slowdown seems to be in the same range as the slowdown reported for self-indexes (Russo *et al.*, 2009). Self-indexes operate completely in main memory, whereas PTPan utilizes slower secondary storage. Nevertheless, compared with self-indexes PTPan shows a slowdown of factor 33 not until deeper searches.

PTPan provides a sustainable nucleic acid sequence data (DNA/RNA) index. This has been demonstrated by constructing PTPan for SSUParc_104_SILVA database with its 1.45 billion nucleotides on common desktop computers, equipped with various amounts of main memory (Section 4.2). Hence, PTPan construction is not limited by the amount of data at the price of increasing construction time. For long sequences or sequence stretches, such as genomes, the current implementation has only one strong limitation: each individual sequence entry of a given dataset has to entirely fit into main memory during construction.

6 CONCLUSION

PTPan is a space-efficient, persistent nucleic acid sequence data index. With suffix trees on secondary storage as core structure, it combines partitioning, truncation and a new stream-compression technique. It performs well for approximate oligonucleotide string matching, probe design and similarity search functions, all taking indels into account. Even largest gene sequence databases can be searched efficiently on common desktop computers, now and in the future, due to PTPan's superior sustainability. As future core component of the worldwide used software package ARB, complementing or even replacing the ARB PT-Server, it will offer thousands of users advanced functionality with significantly reduced hardware requirements.

Currently, the only PTPan trade-off is the slower, though still, reasonable query time. To overcome this constraint facing the continuous data flood, the efficient usage of multicore architectures by parallelization could be a promising continuative approach.

ACKNOWLEDGMENT

We would like to thank Jörg Böhnel for development support and Kai Christian Bader for critically reading the manuscript.

Funding: Bayerische Forschungsstiftung (AZ 767-07).

Conflict of Interest: none declared.

REFERENCES

Abouelhoda,M.I. *et al.* (2004) Replacing suffix trees with enhanced suffix arrays. *J. Dis. Algorithms*, **2**, 53–86.

Amann,R. and Fuchs,B.M. (2008) Single-cell identification in microbial communities by improved fluorescence in situ hybridization techniques. *Nat. Rev. Microbiol.*, **6**, 339–348.

Amaral-Zettler,L.A. *et al.* (2009) A method for studying protistan diversity using massively parallel sequencing of V9 hypervariable regions of small-subunit ribosomal RNA genes. *PLoS One*, **4**, e6372.

Bader,K.C. *et al.* (2011) Comprehensive and relaxed search for oligonucleotide signatures in hierarchically-clustered sequence datasets. *Bioinformatics*, **26**, 1546–1554.

Barsky,M. *et al.* (2010) A survey of practical algorithms for suffix tree construction in external memory. *Softwr. Pract. Exp.*, **40**, 965–988.

DeSantis,T.Z. *et al.* (2006) Greengenes, a chimera-checked 16S rRNA gene database and workbench compatible with ARB. *Appl. Environ. Microbiol.*, **72**, 5069–5072.

Hodges,C.P. (2003) *Distributed Data Structures for Efficient Molecular Sequence Analysis*. Master's Thesis, Technische Universität München, Fakultät für Informatik.

Huffman,D. (1952) A method for the construction of minimum-redundancy codes. *Proc. I.R.E.*, **11**, 91–99.

Kim,J.M. *et al.* (2010) Analysis of the fine-scale population structure of 'Candidatus accumulibacter phosphatis' in enhanced biological phosphorus removal sludge, using fluorescence in situ hybridization and flow cytometric sorting. *Appl. Environ. Microbiol.*, **76**, 3825–3835.

Kumar,Y. *et al.* (2005) Graphical representation of ribosomal RNA probe accessibility data using ARB software package. *BMC Bioinformatics*, **6**, 61.

Kurtz,S. *et al.* (2004) Versatile and open software for comparing large genomes. *Genome Biol.*, **5**, R12.

Li,H. and Durbin,R. (2009) Fast and accurate short read alignment with Burrows-Wheeler transform. *Bioinformatics*, **25**, 1754–1760.

Loy,A. *et al.* (2008) probeCheck - a central resource for evaluating oligonucleotide probe coverage and specificity. *Environ. Microbiol.*, **10**, 2894–2898.

Ludwig,W. *et al.* (2004) ARB: a software environment for sequence data. *Nucleic Acids Res.*, **32**, 1363–1371.

McIlroy,S.J. *et al.* (2011) Non-target sites with single nucleotide insertions or deletions are frequently found in 16S rRNA sequences and can lead to false positives in fluorescence in situ hybridization (FISH). *Environ. Microbiol.*, **13**, 33–47.

Moffat,A. *et al.* (2009) Reducing space requirements for disk resident suffix arrays . In *Database Systems for Advanced Applications*, Vol. 5463/2009 of *Lecture Notes in Computer Science*. Springer, Berlin/Heidelberg, pp. 730–744.

Ohlebusch,E. *et al.* (2010). CST++. In Chavez,E. and Lonardi,S. (eds) *String Processing and Information Retrieval*, Vol. 6393 of *Lecture Notes in Computer Science*. Springer, Berlin/Heidelberg, pp. 322–333.

Phillippy,A.M. *et al.* (2007) Comprehensive DNA signature discovery and validation. *PLoS Comput. Biol.*, **3**, e98.

Pruesse,E. *et al.* (2007) SILVA: a comprehensive online resource for quality checked and aligned ribosomal RNA sequence data compatible with ARB. *Nucleic Acids Res.*, **35**, 7188–7196.

Russo,L.M.S. *et al.* (2009) Approximate string matching with compressed indexes. *Algorithms*, **2**, 1105–1136.

Schönmann,S. *et al.* (2009) 16S rRNA gene-based phylogenetic microarray for simultaneous identification of members of the genus Burkholderia. *Environ. Microbiol.*, **11**, 779–800.

Schulz,M.H. *et al.* (2008) The generalised k-truncated suffix tree for time-and space-efficient searches in multiple DNA or protein sequences. *Int. J. Bioinformatics Res. Appl.*, **4**, 81–95.

Yilmaz,L.S. *et al.* (2008) Systematic evaluation of single mismatch stability predictors for fluorescence in situ hybridization. *Environ. Microbiol.*, **10**, 2872–2885.