Freie Universität Berlin

CHARITÉ
UNIVERSITÄTSMEDIZIN BERLIN

BLAST-like Local Alignments with RazerS

A Thesis Submitted in Partial Fulfillment of the Requirements for the Degree

Bachelor of Science in Bioinformatics

Hannes Hauswedell

30.07.2009 - 01.10.2009

First Supervisor:	Prof. Dr. Knut Reinert
Second Supervisor:	Prof. Dr. Daniel Huson
Advisor:	David Weese

SEQAN

I hereby affirm in lieu of an oath that I have produced this work all by myself. Ideas taken directly or indirectly from other sources are marked as such. This work has not been shown to any other board of examiners so far and has not been published yet.

I am fully aware of the legal consequences of making a false affirmation.

Place/Date

Signature

Contents

List of Figures

List of Tables

1 Introduction

Modern second-generation sequencing technologies are strongly influencing the field of sequence analysis. Ultra-high throughput sequencing technologies produce vast amounts of DNA reads – without the need for amplification through cloning or PCR. The increased availability of these technologies has spawned a lot of research in the field of metagenomics.

Metagenomics is the comparative analysis of large uncharacterized genetic samples obtained from a common habitat (e.g. soil or sea water). Goals include the research of microbial diversity and evolution in a certain environment or the study of organisms that are not easily cultured on artificial conditions.

The dramatic increase in data, however, presented a challenge to sequencing and assembly software. This challenge was met with a new generation of read mappers that implement different approaches to efficiently map complete short reads against a reference genome ("semi-global" alignment). Among modern read mapping software RazerS (Weese et al., 2009) is one of the most efficient and versatile available. It has a high performance and does not exhibit any of the disadvantages many other read mappers do, including limitations in read lengths (i.a. Zoom, Lin et al. 2008; Soap, Li et al. 2008b), restriction to Hamming-distance scoring (i.a. Eland, Cox 2006; Maq, Li et al. 2008a), or the lack of full sensitivity control (i.a. Shrimp, Rumble et al. 2009; Seqmap, Jiang and Wong 2008).

However, read mapping cannot be applied to all sequencing problems in metagenomics. For one, the length of "short reads" produced by sequencing hardware increases ever and with it the probability that reads reach into or even span an intron, effectively preventing semi-global alignment between transcriptome and DNA. In other situations it is advantageous to search directly on a protein database and not on DNA. Therefore traditional local alignment search tools, like BLAST (Altschul et al., 1997) are still common in metagenomics.

BLAST stands for Basic Local Alignment Search Tool, and is the de-facto standard for heuristic local alignments. It is often referred to as the most widely employed bioinformatics tool ever, the paper of the first version (Altschul et al., 1990) is the most highly cited paper of the 1990s. With version 2 (Altschul et al., 1997) there have been substantial improvements to performance and amongst other features, support for aligning with gaps was added. Despite its lower speed compared to read mappers, this makes BLAST an attractive choice, even in the field of metagenomics. For instance MEGAN (Huson et al., 2007) – an advanced graphical desktop-application for visualizing and analyzing metagenomic datasets – operates on BLAST output.

The obvious question is, can an alternative to BLAST be devised that performs better, especially or at least for a specific use case like metagenomics? This work is to show that it is principally possible to create a local alignment search tool that is faster than BLAST and on the other hand compatible to it in many ways.

To this end, we will create a modified version of RazerS, called RazerBlastS that – like RazerS – builds upon the high-performance, generic C++-library SeqAn (Döring et al., 2008).

2 Methods

Adapting RazerS to the task at hand, involves many different steps, of which the algorithmic background will be discussed in this chapter.

Read mapping can usually be divided into two phases, filtration and verification. In filtration, candidate regions on the genome are identified by a low complexity algorithm. These regions are then examined in a more expensive verification step that decides about its status as a match. We will apply this functional devision of the algorithm to BLAST as well and compare both phases in BLAST, RazerS and RazerBlastS.

As both the blastn-mode and the blastx-mode are to be implemented, RazerS needs to be adapted to support amino acids in addition to nucleotides. Fortunately, SeqAn is designed in a generic way, based on template-subclassing, so most algorithms are agnostic of the alphabet in use and little changes have to be made in that respect. However, initially the sequences will have be to translated, which is briefly described here.

2.1 Translation

In blastx-operation, the database is a protein-database, while the query is DNA. Hence the query sequences need to be translated from nucleotide to amino acid alphabet, which is done with the help of a codon translation table (see fig.2.1). In RazerBlastS this was implemented as a 4^3-array that allows constant-time conversion of a DNA/RNA-triplet. Other translation tables beside the *Universal Code* exist – and should be rather easy to add to RazerBlastS – but have not been included in this first implementation.

Since DNA/RNA to amino acid conversion is three-letter to one-letter, there are three different possible translations, called frames (fig.2.2). As usually both genome strands are analyzed (`-S 3`), we get six amino acid strings for every input sequence.

2.2 Filtration

2.2.1 BLAST

The well known BLAST algorithm identifies its alignment candidate regions in the following steps:

1. generate the list of all k-letter words in the query string (where k is set by the parameter `-W` and defaults to 11 for nucleotides and 3 for amino acids, see tab.2.1)
2. for each k-letter word, also add the *word neighborhood*, i.e. all k-letter words with high similarity to the list (only for amino acids)
3. build a tree-like structure from the word-list
4. scan the database for exact hits using the tree

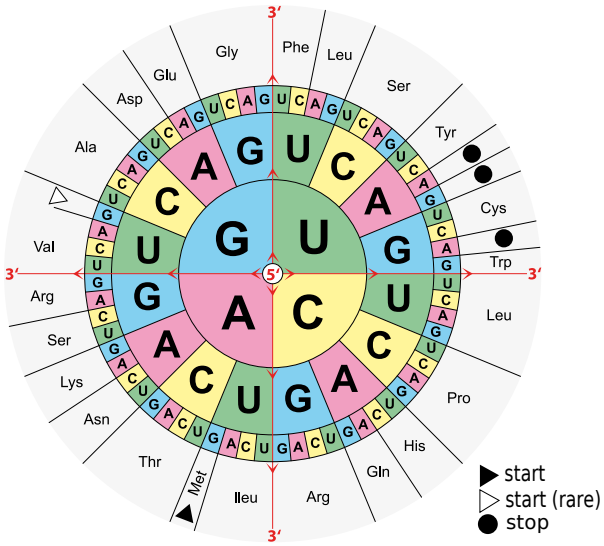

Figure 2.1: Standard genetic amino acid code
©Wikimedia Commons / cc-by-sa

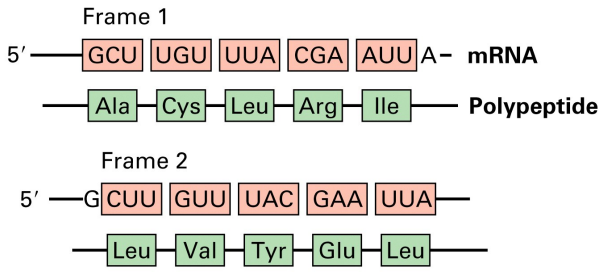

Figure 2.2: Translation in frames
(from Lodish, 5.Ed., ©W.H. Freeman & Co)

2.2.2 RazerS

RazerS uses the Swift Algorithm (Rasmussen et al., 2006) for filtration. The implementation is based on the q-gram lemma (Owolabi and McGregor, 1988; Jokinen and Ukkonen, 1991) and works with an index of gapped or ungapped shapes (Burkhardt and Kärkkäinen, 2003). Gapped shapes result in a higher sensitivity while maintaining equal specificity.

The q-gram lemma states that two sequences with length n and Hamming or edit distance k share at least

$$t = n + 1 - (k + 1)q$$

q-grams, where a q-gram is a common substring of length q. The original RazerS paper (Weese et al., 2009) explains further:

> "For any read $r \in R$ each dot plot parallelogram of dimension $|r| \times (k + 1)$ with at least t q-hits contains a potential match. Instead of counting q-hits for each possible parallelogram separately, it suffices to count them in overlapping $|r| \times w$ parallelograms with $w > k + 1$ and an overlap of k, as every $|r| \times (k + 1)$ parallelogram is contained in one $|r| \times w$ parallelogram, see [fig.2.3].
>
> If for $i, j \in \mathbb{N}$ holds $r[i..i + q - 1] = G[j..j + q - 1]$, the corresponding q-hit is covered by the diagonal $j - i$. For an overlap of k and $w = d + k$ the $|r| \times w$ parallelograms begin at diagonals $0, d, 2d, ...$ If d is a power of 2, the parallelograms containing a q-hit can efficiently be determined by bit-shifting $j - i$.
>
> q-hits are determined by searching overlapping q-grams $G[j..j + q - 1], j = 1, ..., |G| - q + 1$ in a q- gram index of all overlapping q-grams of sequences in $\mathbb{R}$. Only a small number of counters is needed per read, when sliding the q-gram over G. As every $|r| \times w$ parallelogram spans at most $|r| + w - 1$ letters of G, a parallelogram counter can be re-used after $|r| - w - q$ sliding steps. Before re-using a counter, the associated parallelogram is verified if the counter has reached threshold t.
>
> The Swift approach can also be used with gapped shapes Q using a Q-gram index and replacing each q in the formulas above by $span(Q)$."

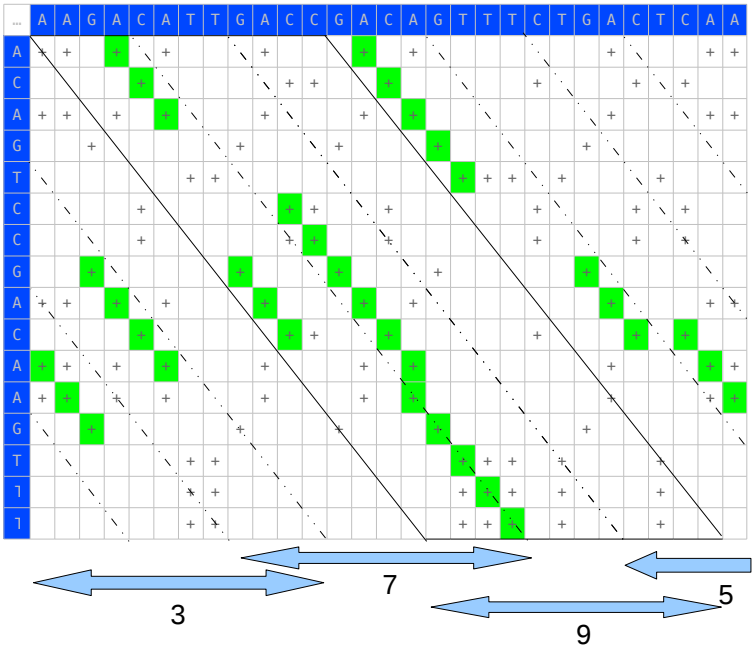

Figure 2.3: Q-Gram counting in overlapping parallelograms ($Q = 3$)

Filter parameters RazerS comes with a parameter chooser that automatically computes the shape Q and the threshold t to achieve a given minimum sensitivity with optimal running time. This is done with a database of pre-computed shape/threshold combinations that were generated for read lengths 24 to 100 (for read length > 100 values are extrapolated), error rates up to 10% and based on the Illumina error profile (Dohm et al., 2008). The accuracy of the parameter chooser was verified on simulated and real data. For more information see Weese et al. (2009).

Automatic parameter choosing can be deactivated by specifying `--shape` and/or `--threshold` manually.

2.2.3 RazerBlastS

RazerBlastS uses the RazerS filter with only a little modification. Since we do not aim at semi-global alignments, a parameter was added to reduce the size of the parallelogram. The factor defaults to 0.4, but can be set manually with `--read-length-adjust`.

For blastn-mode the automatic shape and threshold calculation was retained, with the possibility of either overwriting it the through the mentioned RazerS-parameters(`--shape` and `--threshold`) or through the BLAST-parameter `-W` (`-W x` sets an ungapped shape of length x and a threshold of 2 for gapped alignments and 1 for ungapped alignments). This was done, because automatic parameter calculation is major advantage of the RazerS filter compared to BLAST. A more traditional blastn-like behavior can be achieved with `-W 11`.

For blastx-mode RazerBlastS cannot use automatic parameter calculation, since the mechanisms where optimized for nucleotide reads. To be close to BLAST's default behavior RazerBlastS defaults to `-W 3` in blastx-mode. However it is still possible to use gapped shapes or larger thresholds by specifying them manually (with the aforementioned RazerS-parameters).

	`blastall`				RazerBlastS			
	blastn		blastx		blastn		blastx	
	gaps	w/o gaps	gaps	w/o gaps	gaps	w/o gaps	gaps	w/o gaps
pattern length	11		3		*auto*		3	
pattern count	1	2	1		*auto*	2	1	

Table 2.1: Default filter parameters
pattern length is word length for BLAST and shape length for RazerBlastS
pattern count is two-hit/one-hit extension for BLAST and threshold for RazerBlastS

2.3 Verification

2.3.1 BLAST

Extension Between version 1 of BLAST(Altschul et al., 1990) and version 2 (Altschul et al., 1997) the verification step changed significantly. In BLAST2 it follows the following scheme:

1. for every hit check if there is another non-overlapping hit on the diagonal within distance A (the word neighborhood threshold in filtration was lowered to increase the likelihood for this)
2. if yes, combine these into a segment and extend it in traditional BLAST-manner (without gaps, as long the score does not fall more tha X_u below the best score yet found)
3. if the resulting HSP has a score $> S_g$ choose its best-scoring substring of length $|HSP| - 11$ (or the central character-pair if $|HSP| < 11$) and use that as the seed for a gapped alignment
4. this gapped alignment is also realized via an X-Drop approach, where the score of the alignment must not fall more than X_g below the best score yet found

Significance evaluation If a match is found with this method, its score S is calculated and then normalized:

$$S' = \frac{\lambda S - \ln K}{\ln 2} \qquad \text{(Eq.1)}$$

Where λ and K are constants specific to the scoring scheme (Altschul and Gish, 1996). The normalized score S' is to said to be expressed in *bits*. Based on S' we can then calculate the E-Value:

$$E = nm2^{-S'} \qquad \text{(Eq.2)}$$

The E-Value is the number of sequences with score S expected under random circumstances and therefore a direct indicator for significance. The E-Value-Threshold (parameter `-e`) finally decides whether an alignment is included in the output or not.

2.3.2 RazerS

RazerS verifies gapped alignments with Myers Bitvector algorithm (Myers, 1999). The Bitvector algorithm takes advantage of fast binary operations to calculate (sub-)strings with minimal edit distance, resulting in the best semi-global alignment between read and genome.

For ungapped alignments, each diagonal is scored character-by-character, as long as fewer than k mismatches occur.

2.3.3 RazerBlastS

Since we need real local alignments and arbitrary scoring schemes, Myers Bitvector algorithm was not usable. To not fall short of the desired sensitivity, we decided to use the exact Gotoh algorithm (Gotoh, 1982), which is based on the Smith-Waterman algorithm (Smith and Waterman, 1981), but performs affine gap costs calculation and stays in $O(n^2)$ running time.

Unfortunately a banded implementation of the algorithm was not available for local alignments, so not only the parallelograms, but entire rectangles are verified (see fig.2.4). (In our case reducing the DP-Matrix to the parallelogram as band would not have wasted information, since the filter already processes the database in overlapping parallelograms).

For ungapped alignments we score all diagonals and choose the best overall substring. Diagonals are scored similar to Smith-Waterman (scores < 0 are treated as $= 0$; the best substring of one diagonal ends at the maximum score and starts at the last zero-score before the maximum).

Both approaches work with user selectable match-, mismatch-, gap-open- and gap-extension-scores/penalties, as long as statistical parameters are available for the combination (see below). For blastx Blosum62-based scoring is supported as well.

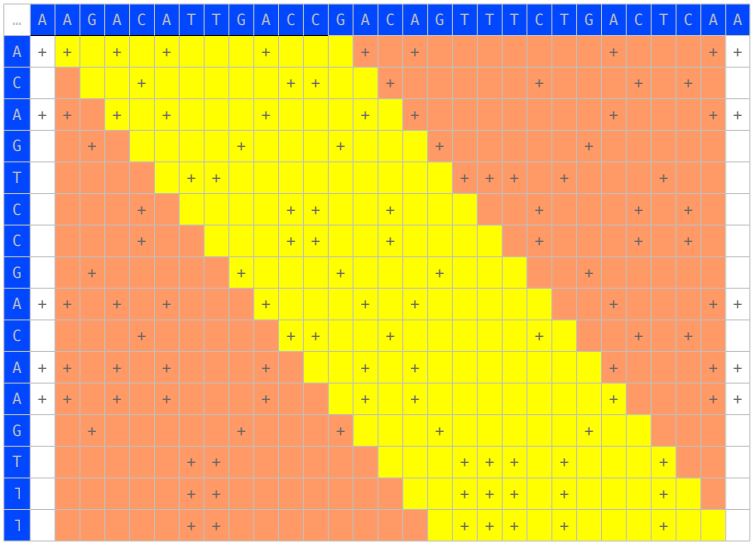

Figure 2.4: With a banded implementation only the yellow area would have to be computed. Right now we are also computing the orange area, which significantly increases the number of operations

Significance evaluation For BLAST-like statistical evaluation, statistical information and tables from `algo/blast/core/blast_stat.c` [1] were imported and adapted for RazerBlastS. Among these are precalculated "Karlin-Altschul-Values" (Altschul and Gish, 1996) – λ, K and H – for different nucleotide and protein scoring schemes.

Based on these values RazerBlastS implements normalization (Eq.1) and E-Value calculation (Eq.2). Unfortunately it was not possible to implement edge-effect correction (Altschul, 1997), yet. Therefore E-Values are expected to be slightly larger in RazerBlastS compared to BLAST.

[1] Source-Code of: ncbi-toolkit-2009.03.01 (containing BLAST v2.2.19)

3 Implementation

The main BLAST-tool, `blastall` has a different operation modes for different tasks (selectable with the `-p` parameter). While the initial focus of RazerBlastS was on Nucleotide-BLAST (blastn), it was decided later to also implement Nucleotide-vs-Protein-BLAST (blastx), as it is commonly used in metagenomics.

The algorithms discussed previously were adapted mostly from existing RazerS and SeqAn-code. In this chapter we will give an overview over the application interface, as well as input and output formats developed for or available in RazerBlastS.

3.1 Interface

RazerBlastS supports two kind of parameters, BLAST-style parameters (tbl.3.1) and RazerS-style parameters (tbl.3.2). To be as close as possible to BLAST's parameter handling, most of the frequently used options were implemented and RazerS options were replaced where possible. Those that were not replaced and are still applicable to RazerBlastS were retained in their *"long format"*, so as not to conflict with BLAST's one-letter parameters. Altogether only a subset of all BLAST options is available, however they should suffice for the task at hand.

`-p blastn|blastx`	program (operation mode)
`-r x`	score for a match[1]
`-q x`	penalty for a mismatch[1]
`-G x`	penalty for gap-existance
`-E x`	penalty for gap-extension
`-g T|F`	gapped alignments (true|false)
`-S 1|2|3`	query strands (top|bottom|both)
`-e x`	e-Value Threshold
`-W x`	word size / explicit shape (filter)
`-i filename`	input file (queries)
`-d filename`	input file (database)
`-o filename`	output file

Table 3.1: Supported BLAST parameters
[1]blastn-only

Default values for all options were chosen to be BLAST-like, with some notable exceptions, as noted above (tbl.2.1). Of all parameters only `-p`, `-i`, `-d` and `-o` are required, so when using the mentioned subset of options it should be possible to use RazerBlastS as a drop-in-replacement for `blastall`.

`--shape BITSTRING`	shape (deactivate param.chooser)
`--threshold x`	minimum k-mer threshold
`--recognition-rate x`	minimum sensitivity
`--overabundance-cut x`	k-mer overabundance cut ratio
`--repeat-length x`	simple-repeat length threshold
`--taboo-length x`	taboo length
`--read-length-adjust x`	adjust the read length given to filter
`--percent-identity x`	error tolerance of the filter
`--low-memory`	optimize for low memory (smaller shapes)
`--(v)verbose`	increased verbosity

Table 3.2: Supported RazerS parameters

3.2 Input

	Query	Database
`blastall`	FASTA, ASN.1	binary
RazerS	FASTA, FASTQ, Genbank	

Table 3.3: Input file formats

FASTA The FASTA file format was developed at the NCBI with the FASTA local alignment software. While the FASTA program was superseded by BLAST for all use cases, the original FASTA file format is still a de-facto standard for nucleotide and protein sequences. It is supported natively by BLAST for query files and for both query and database by RazerS.

FASTQ A lot of short-read sequencing data for queries is available in FASTQ, a format that, in addition to FASTA-like sequence information, also contains read quality estimates for all characters. These quality estimates are based on the properties of the sequencing hardware that isn't equally accurate on every position. FASTQ is supported natively by RazerS and can be converted to FASTA for use with BLAST by tools like SAK (*"Swiss Army Knife"*), a small program based SeqAn for format conversion and sequence extraction.

BLAST's binary databases For performance reasons BLAST uses preformatted and indexed binary databases. They are generated from FASTA or ASN.1 formats with the help of `formatdb`, another tool from the NCBI-toolkit. RazerS has no support for these databases, so the reverse process (conversion from binary format to FASTA) is necessary for databases like `nr`. This is possible with `fastacmd`, which is also included in the NCBI-toolkit.

As `blastall` takes the FASTA-file as formal filename-argument (although it then reads the binary files) command-line syntax is identical to RazerBlastS (see tab.3.1).

3.3 Output

BLAST supports a variety of output formats. Although there are certain advantages to more machine readable formats like XML, we chose to implement the traditional BLAST report. It is still the default format and very common. MEGAN also processes traditional BLAST reports, so we will be able to verify a certain degree of formal correctness by feeding our output to MEGAN.

4 Results

In this chapter we will perform tests of RazerBlastS on real data and compare it with BLAST. It should be noted that these are test-datasets and in some aspects not typical for metagenomics. The same is true for the choice of parameters that was sometimes made to prove certain default behavior and was limited by hardware constraints in other situations.

All test were conducted on Intel Xeon machines with 3.2Ghz, 2-4 CPU-Cores and 6GiB Ram. The operating system used was Debian GNU/Linux, version 5.0.3 ("Lenny"), a free UNIX-like operating system [1]. As support for threading is not stable in RazerS, yet, the single-threaded versions of BLAST and RazerBlastS were used in all experiments.

For each experiment, quality (and quantity) of results are compared, as well as run-time and memory usage. The run-time was measured with the UNIX time command; memory usage is the peak heap usage measured by pre-loading the memusage-library (stack memory usage is negligible on both programs).

No work was yet done on synchronizing the behavior of the repeat screeners, so they were deactivated (`-F F`) on all test-runs so as not to impede comparability analysis.

4.1 Experiment 1

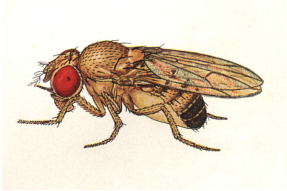

Figure 4.1: D. Melanogaster (male)
©Genome Informatics Lab of Indiana University

	Query	Genome
Species	Drosophila Melanogaster	
Description	Transcriptome Study	2R-Chromosome
Size(in bases)	11.5M	21.6M
Source	NCBI short read archive[2]	FlyBase[3]

Table 4.1: Dataset (experiment 1)

In the first experiment we will compare the blastn-modes of RazerBlastS and BLAST. Query and genome sequences were selected from the Drosophila Melanogaster organism (see tbl.4.1).

Parameters BLAST was invoked with the following parameters:

```
blastall -p BLASTN -r 2 -e 0.1 -i SRR005468.fasta \
         -d dmel-2R-chromosome-r5.21.fasta -o out -F F
```

[1] http://debian.org
[2] http://www.ncbi.nlm.nih.gov/sites/entrez?db=sra&term=SRX001435&report=full
[3] ftp://ftp.flybase.net/genomes/Drosophila_melanogaster/current/fasta/dmel-2R-chromosome-r5.21.fasta.gz

RazerBlastS was invoked with similar arguments:

```
razerblasts -p BLASTN -r 2 -e 0.3 -i SRR005468.fasta \
            -d dmel-2R-chromosome-r5.21.fasta -o out --vverbose
```

The difference in *E*-Value threshold is due to the aforementioned missing edge-effect correction in RazerBlastS that results in slightly higher *E*-Values. Tests have shown that these two thresholds produce similar results in blastn-Mode.

Results The output of both program runs was parsed with a combination of UNIX tools, including GNU AWK[4] and sort, to produce a formatted list of the best alignments (sorted by *E*-Value). The best 50 alignments produced by both tools are identical, with minor differences in sorting (see tbl.4.2 for the first few). Appendix A.1 (p.22) contains each program's full output for the best alignment.

RazerBlastS			Blast		
E-Value	*bits*	ReadId	*E*-Value	*bits*	ReadId
3e-148	522.5	SRR005468.22871	1e-148	522	SRR005468.22871
5e-146	515.2	SRR005468.16600	1e-146	515	SRR005468.16600
5e-146	515.2	SRR005468.27361	1e-146	515	SRR005468.27361
3e-142	502.6	SRR005468.53331	1e-142	502	SRR005468.53331
5e-140	495.4	SRR005468.28974	1e-140	495	SRR005468.28974
2e-137	486.4	SRR005468.15811	1e-138	488	SRR005468.25265
2e-137	486.4	SRR005468.25265	1e-137	484	SRR005468.26229
8e-137	484.6	SRR005468.26229	1e-137	486	SRR005468.15811
1e-134	477.4	SRR005468.26112	1e-134	477	SRR005468.26112
1e-133	473.8	SRR005468.56215	1e-133	471	SRR005468.39513
5e-133	472.0	SRR005468.39513	1e-133	473	SRR005468.56215
7e-131	464.8	SRR005468.53536	1e-131	464	SRR005468.53536
8e-131	464.8	SRR005468.56043	1e-131	464	SRR005468.56043
8e-130	461.1	SRR005468.45656	1e-130	461	SRR005468.22808
9e-130	461.1	SRR005468.22808	1e-130	461	SRR005468.45656
3e-129	459.3	SRR005468.23891	1e-129	457	SRR005468.40897
3e-129	459.3	SRR005468.25855	1e-129	459	SRR005468.23891
1e-128	457.5	SRR005468.40897	1e-129	459	SRR005468.25855
4e-127	452.1	SRR005468.23669	1e-127	452	SRR005468.23669
5e-127	452.1	SRR005468.44427	1e-127	452	SRR005468.44427

Table 4.2: The best alignments of both runs (experiment 1)

Performance Given values are the best out of three identical runs (variance in run-time was < 30s for both programs). RazerBlastS finishes in 63% of BLAST's run-time, but requires significantly more memory.

	RazerBlastS	Blast
run-time	7m18s	11m34s
memory	625MiB	32MiB

Table 4.3: Performance (experiment 1)

[4] http://www.gnu.org/software/gawk/

4.2 Experiment 2

Figure 4.2: Eucalyptus Grandis
public domain work available from Wikimedia Commons

	Query	Genome
Species	Eucalyptus Grandis	*miscellaneous*
Description	Transcriptome	Subset of `nr`
Size(in bases)	1.9M	$\sim$ 250M
Source	NCBI short read archive[5]	NCBI BLAST

Table 4.4: Dataset (experiment 2)

In the second experiment the blastx capabilities are to be compared. For performance reasons, a small-sized random subset of the BLAST `nr` database was extracted and used. The query consists of short reads (SRR001658) from Eucalyptus Grandis, whose genome is not yet characterized.

Parameters The programs were invoked with the following parameters:

```
blastall -p BLASTX -e 0.1 -i SRR001658.fasta -d small_nr.fasta -o out \
         -F F
```

and:

```
razerblasts -p BLASTX -e 0.1 -i SRR001658.fastq -d small_nr.fasta -o out \
            --vverbose
```

On BLASTX test-runs against `nr` the missing edge effect correction seemed less significant and RazerBlastS produced more results, so the *E*-Value thresholds in this experiment are the same.

Performance The BLAST run finished after 3h34m, while the RazerBlastS run had to be interrupted after 4h, because up to that point it had reached only approximately 3% progress. Evidently the default filter settings for RazerBlastS are too slow.

[5] http://www.ncbi.nlm.nih.gov/sites/entrez?db=sra&term=SRX000427

4.3 Experiment 3

4.3.a

	Query	Genome
Species	Eucalyptus Grandis	*miscellaneous*
Description	Transcriptome	`nr` (complete)
Size(in bases)	313	$\sim$ 5G
Source	NCBI short read archive	NCBI BLAST

Table 4.5: Dataset (experiment 3)

To produce alignments in the given amount of time we decided to reduce the number of reads to a total of three reads from the above sample (reads 6, 23 and 38 from SRR001658). The intent of a query that small, was to be able to search the complete `nr`-database.

Because RazerS loads the complete database into main memory, the `nr`-database was split into four and searched in sequential runs. The results were concatenated. For comparability the same was done with BLAST in addition to a regular BLAST run.

Parameters The programs were invoked with the following parameters (where * stands for 0-3):

```
blastall -p BLASTX -e 0.1 -i reads.fasta -d nr.fasta -o out  -F F
```

```
blastall -p BLASTX -e 0.1 -i reads.fasta -d nr0*.fasta -o out_0*  -F F -I
```

```
razerblasts -p BLASTX -e 0.1 -i reads.fasta -d nr0*.fasta -o out_0*
```

Results Most of the best alignments of each program are identical, with minor differences in sorting (see tbl.4.6). The best alignment of each program is printed completely in Appendix A.2 (p.23).

RazerBlastS				Blast			
E-Value	*bits*	ReadNr	gi	*E*-Value	*bits*	ReadNr	gi
3e-18	68.9	6	118482830	2e-10	68.9	6	118482830
5e-18	68.9	6	224065405	2e-10	68.9	6	224065405
8e-18	67.4	6	1172597	5e-10	67.4	6	1172597
5e-17	64.7	6	225470816	2e-09	65.5	23	255562096
2e-16	62.8	6	115488282	2e-09	64.7	23	147789708
2e-16	64.7	23	147789708	3e-09	64.7	23	157355950
2e-16	65.5	23	255562096	3e-09	64.7	23	225429102
3e-16	64.3	23	157329195	3e-09	64.7	6	225470816
3e-16	64.3	23	225438223	4e-09	64.3	23	147801370
3e-16	64.3	23	225467504	4e-09	64.3	23	224067152
3e-16	64.7	23	157355950	4e-09	64.3	23	225438223
4e-16	62.0	6	224132228	4e-09	64.3	23	225467504
4e-16	64.3	23	147801370	4e-09	64.3	23	157329195
4e-16	64.3	23	224067152	9e-09	63.2	23	18424995

Table 4.6: The best alignments of both runs (experiment 3)

The amount of alignments produced differ by magnitudes; BLAST produces 46 alignments between the three reads and the complete `nr` database, while RazerBlastS produces 1236026 alignments. This is due to the curious fact that in this experiment *E*-Values calculated by RazerBlastS are significantly smaller than BLAST's (while *bits*-score are still identical or very close).

Filtering the results through GNU awk with a manual *bits*-score threshold of the lowest *bits*-score produced by BLAST ($= 53.5 bits$), yields 45 alignments. The "missing" match is also found by RazerBlastS, but scores 52 *bits* and is aligned a little different (see Appendix A.2). Further investigation would have to clarify the reason for this. The next best RazerBlastS alignment however, scores significantly worse (36.2 *bits*) which strongly suggests that with correct *E*-Value calculation it (and all following alignments) would not have been included.

Performance Table 4.7 contains run-time and memory usage of the program runs, measured as in experiment 1. Note that the different segments of `nr` are those that can be found on the NCBI ftp server [6], which are not equal in size. The last line contains the sum of the individual runs. The sums are just given as an overview over the required resources for the tests, they are not significant on their own, i.e. a single RazerBlastS run on more potent hardware would have used less memory and probably also less time.

RazerBlastS		Blast		Blast (one-run)	
run-time	memory	run-time	memory	run-time	memory
17m28s	2.16GiB	39s	2.7MiB		
19m06s	1.87GiB	44s	2.7MiB		
19m48s	2.00GiB	53s	2.7MiB		
10m22s	1.11GiB	25s	2.7MiB		
65m42s	7.14GiB	2m41s	10.8MiB	2m46s	2.7MiB

Table 4.7: Performance (experiment 3)

4.3.b

Using the same dataset, as in part **a** of this experiment, we will now measure the influence of different filter parameters on amount and quality of results, as well as run-time and memory usage. We will use only the fourth ("03") segment of `nr` and also check if the eight alignments found by BLAST are contained in the output. Note, however, that containing these eight alignments in itself does not warrant for sensitivity in general, since these alignments are very high scoring.

RazerBlastS is run with the same parameters as above, additionally `--shape`, `--threshold` and `--verbose` are specified (the latter to receive filter statistics).

The results (tbl.4.8) show significant speed increases when modifying shape and/or threshold, while all runs, but one, include the results BLAST gives. Even the runs producing comparably few alignments, produce many alignments that score the exact *E*-Value threshold of 0.1, showing that these filter settings are principally not too "strict" for the desired statistical significance. However, as noted above, it is not possible to make general assumptions about sensitivity compared to BLAST, based on just this set of data.

[6] `ftp://ftp.ncbi.nlm.nih.gov/blast/db`

shape	threshold	# alignments	# of ali. > 50 *bits*	run-time	memory
`111`	2	193174	8	9m24s	1.1GiB
`111`	3	23918	8	1m39s	1.1GiB
`111`	4	8411	8	40ss	1.1GiB
`111`	5	3069	8	38s	1.1GiB
`1101`	2	52951	8	6m04s	1.1GiB
`1101`	3	14647	8	59s	1.1GiB
`1101`	4	4227	8	32s	1.1GiB
`1111`	2	10166	8	56s	1.1GiB
`1111`	3	2338	8	33s	1.1GiB
`1111`	4	449	7	35s	1.1GiB
`11011`	2	4854	8	43s	1.1GiB
`11011`	3	654	8	39s	1.1GiB

Table 4.8: Results & Performance (experiment 3b)

4.4 Experiment 4

In the fourth experiment MEGAN shall be used to compare local alignment runs. In part **a** BLAST and RazerBlastS will generate data that we will visualize and analyze with the help of MEGAN in part **b** of the experiment.

4.4.a

The dataset used, is that of experiment 2 (tbl.4.5, p.12). Both BLAST and RazerBlastS will be run also in their ungapped mode (`-g F` for both). The default filter settings for RazerBlastS (tbl.2.1) will not be used, as they have shown to be too slow. Among those options selected, are also rather large shapes and thresholds as we expect MEGAN use cases where it is desirable to sacrifice a certain degree of sensitivity for speed.

	BLAST		
	# alignments	run-time	memory
gapped	367076	3h34m	12.4MiB
ungapped	161613	3h19m	5.2MiB

Table 4.9: Performance BLAST (experiment 4)

	RazerBlastS					
	shape	threshold	# alignments	# ali. > 34*bits*	run-time	memory
gapped	`1101`	4	19881971	346467	9h32m	2.27GiB
	`101101`	8	648624	119276	2h50m	0.55GiB
	`1011101`	3	2584809	191027	1h20m	0.78GiB
	`101101101`	1	6667956	257043	1h47m	2.79GiB
ungapped	`101101`	8	563870	156541	3h03m	0.51GiB
	`1011101`	3	2105092	860594	1h38m	0.76GiB
	`101101101`	1	5706524	1269423	2h13m	2.51GiB

Table 4.10: Performance RazerBlastS (experiment 4)

Results As in the previous experiment the RazerBlastS results were filtered for the minimum *bits*-score found in the BLAST-runs. For the gapped alignments, this shows a varying degree of sensitivity compared to BLAST and depending on the choice of filter parameters. Provided that all *bits*-score are calculated correctly, none of the runs achieve equal sensitivity to BLAST. A more in depth analysis, unfortunately not in the scope of this work, will have to show whether this can be reached in an affordable way.

While the best alignments in ungapped mode are also identical to BLAST's ungapped mode, the *bits*-scores differ greatly (the raw scores are identical, though). Since the ungapped mode of BLAST2 is not discussed in the paper (Altschul et al., 1997), likely certain statistical specifics were not correctly implemented in RazerBlastS. This explains the unexpectedly high number of scores $>= 34$ *bits*. Based on these results it is currently not possible to estimate sensitivity in ungapped mode.

Performance The run-time varies strongly between the different runs, but six out of the seven runs are faster than BLAST, even in ungapped mode. Three runs finish in less than half the time, while one of them still produces $\sim 70\%$ of the amount of matches BLAST does. Curiously the run-time of runs in ungapped mode is higher than in gapped mode (for equal parameters). Apparently there is still room for improvement in the implementation.

Memory usage is significantly higher than BLAST, as described earlier. Since it depends both on the length of the shape and the matches found, it varies between the different runs.

4.4.b

File import Importing the output files generated by RazerBlastS into MEGAN revealed two faults in the format implementation that caused misbehavior or loss of information:

1. RazerBlastS does not introduce line-breaks in genome-headers, causing errors when especially long headers are read by MEGAN
2. RazerBlastS creates a read-header for each reading frame, instead of creating one for each read, which leads MEGAN to interpret the reading frames as different reads (apparently MEGAN does not compare the query names)

While these issues could be fixed in the program it was also desirable to fix the files already produced. For the first problem this was achieved with the GNU sed [7], for the second with GNU awk.

With the issues solved, output files were imported into MEGAN without visible errors and converted to MEGAN's own file format, RMA. The default values for settings like minimal *bits*-score and top-percentage were retained. For some runs the read number was manually set as described in the MEGAN manual, because reads with no hits are currently not printed by RazerBlastS.

[7] a small UNIX tool mostly for replacement operations based on regular expressions, http://www.gnu.org/software/sed/

Taxonomic trees Figure 4.3 presents the leaf nodes of all trees in the "heat map" display mode. As the scoring of ungapped alignments has shown to be flawed the results are not discussed here. However an overview can be found, together with the full trees for all runs, in Appendix B (p.25).

Note that only a small subset of the `nr`-database was searched, so the results are not expected to be biologically relevant and the main focus is on comparability with BLAST.

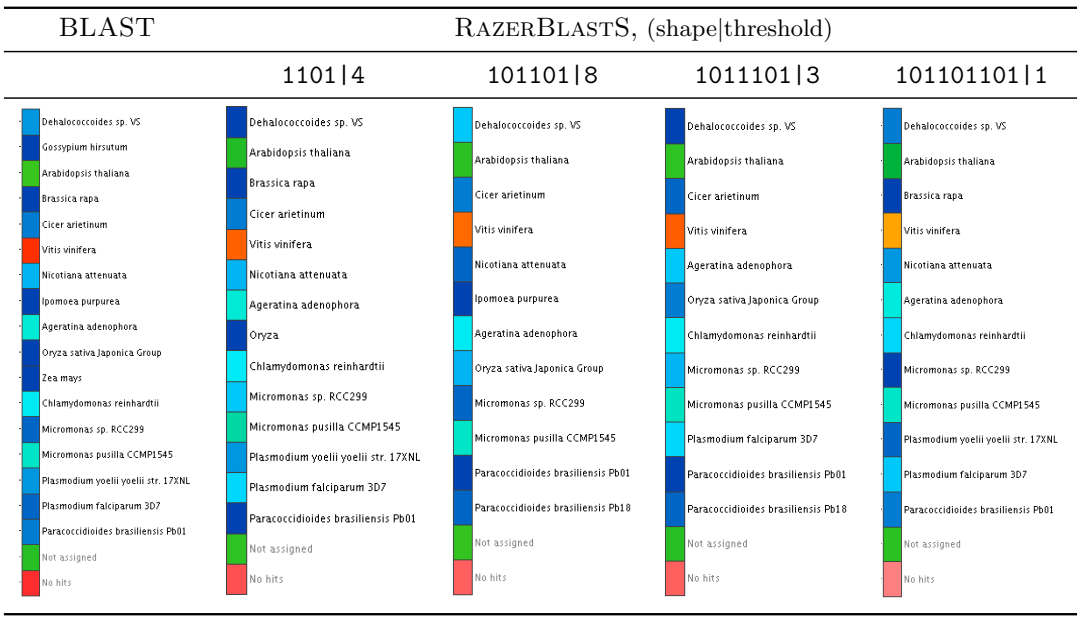

Figure 4.3: Leaf nodes of taxonomic trees generated by MEGAN from gapped alignments

The results for the gapped alignments are promising. As expected, with fewer well scoring matches (tbl.4.10), not all taxa are created in all trees (see also tbl.4.11) and some leaves are missing. However the species found as being most closely related to the query ("Vitis vinefera" and "Arabidopsis thaliana") are found and high scored in all RazerBlastS-runs as well. If it can be assumed that "Oryza" is close to "Oryza sativa Japonica Group" there are also no *false positives*, in the sense that RazerBlastS assumes relatedness where there is none. As visible in Appendix B (p.25) the general composition of the trees is very similar as well.

Further research, beyond the scope of this work, will have to be conducted to compare larger scales of data and runs with comparable sensitivity.

	shape	threshold	# of taxa	# of reads mapped*
BLAST			114	5968
RazerBlastS	1101	4	98	4854
	101101	8	86	4356
	1011101	3	88	4551
	101101101	1	83	2567

Table 4.11: Number of taxa and mapped reads for the taxonomic trees
* total number of reads is 15236

5 Conclusion

The BLAST default output format was implemented successfully with only minor outstanding issues that did not hinder a flawless import by MEGAN.

A BLAST-like interface was developed that enables identical or very similar operation for many regular use cases. This not only eases transition for users acquainted with BLAST, but could permit directly replacing BLAST in automated scripts or more complex setups.

Adapting RazerS to perform local alignment verification was successful. It produces the expected results in all operation modes (nucleotide and protein, with different scoring schemes), however there is still room for improvement in terms of performance. A banded version of the Gotoh algoritm or X-Drop-Extension similar to BLAST could probably yield comparable results in less time. The latter might be a better choice with respect to sensitivity. The implementation of gap-free local alignments needs to be reworked, as it currently presents no run-time advantage over the (more sensitive) gapped alignment.

Statistical evaluation of the alignments works for the most part. The *bits*-scores calculated from the regular alignment scores are identical (or close) to BLAST's in the default gapped mode. This proves that λ- and K-constants were properly imported from BLAST and are correctly selected for the scoring scheme in use. For ungapped alignments the statistical background needs to be reassessed. E-Values are calculated from Eq.2 (p.5), which is only approximately correct, since it does not include modifiers to the search space, like edge effect correction. Therefore E-Values differ slightly from BLAST's, depending on sequence lengths, although the rank of most alignments is identical or similar.

The tests conducted in blastn-mode show very promising results. RazerBlastS finished in less than two thirds of BLAST's run-time and since the parameters were chosen by the parameter-chooser the expected sensitivity is very high. However memory usage in all RazerBlastS-runs exceeds BLAST's by orders of magnitude. This is partly due to the filter index, which cannot be overcome and partly due to RazerBlastS loading the entire genomes into memory, instead of processing a database like BLAST. This restricts use-cases like searching the `nt`-database ($>$ 20GiB uncompressed) to very high-end hardware.

In test-runs of the blastx mode the parameter chooser was not available (it is based on nucleotide statistics) and the default settings chosen for shape and threshold proved to be very slow compared to BLAST. Different manually selected shape and parameter combinations exemplified that RazerBlastS can easily outperform BLAST in blastx-mode as well, however it is difficult to assess the expected sensitivity without a parameter chooser.

If the main issues of high memory usage and lack of a parameter chooser for protein sequences are addressed, Razer(Blast)S and SeqAn present a solid foundation for the development of a high performance local alignment search tool that would outperform BLAST, possibly even on comparable sensitivity. By offering direct control over sensitivity (vs speed) it could be particularly useful for applications in comparative genomics, like MEGAN. With more and more sequence data generated in FASTQ, the recently added support for quality based scoring in RazerS (not yet available in this software), could also provide a considerable advantage over BLAST in regards to specificity.

6 Acknowledgments

During the process of writing this thesis several people have been especially helpful and supportive.

First of all, I would like to thank Prof. Dr. Knut Reinert and Prof. Dr. Daniel Huson for suggesting an interesting and challenging topic in current bioinformatics research. I am most grateful for their willingness to inspire and supervise my work.

Special thanks go to David Weese who was my advisor and helped me with valuable hints and detailed discussions throughout the entire working period.

I want to also pay gratitude to my previous teachers who sparked and nourished my interest in science and mathematics. Their commitment over the years has been highly appreciated.

Furthermore, I want to thank my partner Funda Zillinger and my friends for bearing with me in the last two months. Finally, let me extend my deep gratitude to my parents, for their ongoing empathy and support in general and for making my studies in the current form possible.

References

Altschul, S. F. (1997). Theoretical and computational methods in genome research. *Plenum Press*, page 1–14.

Altschul, S. F. and Gish, W. (1996). Local alignment statistics. *Methods Enzymol*, 266:460–80.

Altschul, S. F., Gish, W., Miller, W., Myers, E. W., and Lipman, D. J. (1990). Basic local alignment search tool. *J Mol Biol*, 215(3):403–10.

Altschul, S. F., Madden, T. L., Schäffer, A. A., Zhang, J., Zhang, Z., Miller, W., and Lipman, D. J. (1997). Gapped blast and psi-blast: a new generation of protein database search programs. *Nucleic Acids Res*, 25(17):3389–402.

Burkhardt, S. and Kärkkäinen, J. (2003). Better filtering with gapped q-grams. *Fundam. Inform.*, 56(1-2):51–70.

Cox, A. J. (2006). Eland: efficient local alignment of nucleotide data. *unpublished*.

Dohm, J. C., Lottaz, C., Borodina, T., and Himmelbauer, H. (2008). Substantial biases in ultra-short read data sets from high-throughput dna sequencing. *Nucleic Acids Res*, 36(16):e105.

Döring, A., Weese, D., Rausch, T., and Reinert, K. (2008). Seqan an efficient, generic c++ library for sequence analysis. *BMC Bioinformatics*, 9:11.

Gotoh, O. (1982). An improved algorithm for matching biological sequences. *J Mol Biol*, 162(3):705–8.

Huson, D. H., Auch, A. F., Qi, J., and Schuster, S. C. (2007). Megan analysis of metagenomic data. *Genome Res*, 17(3):377–86.

Jiang, H. and Wong, W. H. (2008). Seqmap: mapping massive amount of oligonucleotides to the genome. *Bioinformatics*, 24(20):2395–6.

Jokinen, P. and Ukkonen, E. (1991). Two algorithms for approximate string matching in static texts. In *MFCS*, page 240–248.

Li, H., Ruan, J., and Durbin, R. (2008a). Mapping short dna sequencing reads and calling variants using mapping quality scores. *Genome Res*, 18(11):1851–8.

Li, R., Li, Y., Kristiansen, K., and Wang, J. (2008b). Soap: short oligonucleotide alignment program. *Bioinformatics*, 24(5):713–4.

Lin, H., Zhang, Z., Zhang, M. Q., Ma, B., and Li, M. (2008). Zoom! zillions of oligos mapped. *Bioinformatics*, 24(21):2431–7.

Myers, G. (1999). A fast bit-vector algorithm for approximate string matching based on dynamic programming. *J. ACM*, 46(3):395–415.

Owolabi, O. and McGregor, D. R. (1988). Fast approximate string matching. *Softw., Pract. Exper.*, 18(4):387–393.

Rasmussen, K. R., Stoye, J., and Myers, E. W. (2006). Efficient q-gram filters for finding all epsilon-matches over a given length. *J Comput Biol*, 13(2):296–308.

Rumble, S. M., Lacroute, P., Dalca, A. V., Fiume, M., Sidow, A., and Brudno, M. (2009). Shrimp: accurate mapping of short color-space reads. *PLoS Comput Biol*, 5(5):e1000386.

Smith, T. F. and Waterman, M. S. (1981). Identification of common molecular subsequences. *J Mol Biol*, 147(1):195–7.

Weese, D., Emde, A.-K., Rausch, T., Döring, A., and Reinert, K. (2009). Razers–fast read mapping with sensitivity control. *Genome Res*, 19(9):1646–54.

A BLAST & RazerBlastS output samples

A.1 Experiment 1

```
>2R type=chromosome_arm; loc=2R:1..21146708; ID=2R; dbxref=GB:AE013599;
               MD5=1589a9447d4dc94c048aa48ea5b8099d; length=21146708;
               release=r5.21; species=Dmel;
          Length = 21146708

 Score =  522 bits (578), Expect = e-148
 Identities = 291/292 (99%)
 Strand = Plus / Plus


Query: 1       cgccttgtcgtccgctagtatccattccgcctgtcgcttgcgagcctcccagttcttggg 60
               ||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
Sbjct: 6710574 cgccttgtcgtccgctagtatccattccgcctgtcgcttgcgagcctcccagttcttggg 6710633

Query: 61      gagcttcttgcgagcgtcctctgccaccacttcctggtggttgtccgtacgcgcctcctg 120
               ||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
Sbjct: 6710634 gagcttcttgcgagcgtcctctgccaccacttcctggtggttgtccgtacgcgcctcctg 6710693

Query: 121     gcgcttcttgtgtaggtccagcaaacgcgccttgcgctccgccaatttttcggcagccgt 180
               ||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
Sbjct: 6710694 gcgcttcttgtgtaggtccagcaaacgcgccttgcgctccgccaatttttcggcagccgt 6710753

Query: 181     tttcgtcttttccataggtaatttttattaaaaacttgttttgtttttagaaattgtgaa 240
               ||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
Sbjct: 6710754 tttcgtcttttccataggtaatttttattaaaaacttgttttgtttttagaaattgtgaa 6710813

Query: 241     aatgcgttgttttccttttctttgatatcttggtcggttttgggccgcgttgg 292
               |||||||||||||||||||||||||| ||||||||||||||||||||||||
Sbjct: 6710814 aatgcgttgttttccttttctttgatattttggtcggttttgggccgcgttgg 6710865
```

Figure A.1: Best BLAST alignment (experiment1)

```
>2R type=chromosome_arm; loc=2R:1..21146708; ID=2R; dbxref=GB:AE013599;
    MD5=1589a9447d4dc94c048aa48ea5b8099d; length=21146708; release=r5.21; species=Dmel;
          Length = 21146708

 Score = 522.5 bits (579), Expect = 3e-148
 Identities = 291/292 (99%)
 Strand = Plus / Plus

Query: 1       CGCCTTGTCGTCCGCTAGTATCCATTCCGCCTGTCGCTTGCGAGCCTCCCAGTTCTTGGG 60
               ||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
Sbjct: 6710574 CGCCTTGTCGTCCGCTAGTATCCATTCCGCCTGTCGCTTGCGAGCCTCCCAGTTCTTGGG 6710633

Query: 61      GAGCTTCTTGCGAGCGTCCTCTGCCACCACTTCCTGGTGGTTGTCCGTACGCGCCTCCTG 120
               ||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
Sbjct: 6710634 GAGCTTCTTGCGAGCGTCCTCTGCCACCACTTCCTGGTGGTTGTCCGTACGCGCCTCCTG 6710693

Query: 121     GCGCTTCTTGTGTAGGTCCAGCAAACGCGCCTTGCGCTCCGCCAATTTTTCGGCAGCCGT 180
               ||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
Sbjct: 6710694 GCGCTTCTTGTGTAGGTCCAGCAAACGCGCCTTGCGCTCCGCCAATTTTTCGGCAGCCGT 6710753

Query: 181     TTTCGTCTTTTCCATAGGTAATTTTTATTAAAAACTTGTTTTGTTTTTAGAAATTGTGAA 240
               ||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
Sbjct: 6710754 TTTCGTCTTTTCCATAGGTAATTTTTATTAAAAACTTGTTTTGTTTTTAGAAATTGTGAA 6710813

Query: 241     AATGCGTTGTTTTCCTTTCTTTGATATCTTGGTCGGTTTTGGGCCGCGTTGG 292
               |||||||||||||||||||||||||| ||||||||||||||||||||||||
Sbjct: 6710814 AATGCGTTGTTTTCCTTTCTTTGATATTTTGGTCGGTTTTGGGCCGCGTTGG 6710865
```

Figure A.2: Best RazerBlastS alignment (experiment 1)

A.2 Experiment 3

```
>gi|118482830|gb|ABK93331.1| unknown [Populus trichocarpa]
          Length = 47

 Score = 68.9 bits (167), Expect = 2e-10
 Identities = 32/34 (94%), Positives = 33/34 (97%)
 Frame = -2

Query: 104 RSFLSQKGGSSDKRKMEEQRPKEHRPKANENKPV 3
           RSFLSQKGGSSDKRKMEEQ+PKE RPKANENKPV
Sbjct: 11  RSFLSQKGGSSDKRKMEEQKPKEQRPKANENKPV 44
```

Figure A.3: Best BLAST alignment (experiment 3)

```
>gi|118482830|gb|ABK93331.1| unknown [Populus trichocarpa]
          Length = 47

 Score = 68.9 bits (167), Expect = 3e-18
 Identities = 32/34 (94%), Positives = 33/34(97.1%)
 Frame = -2

Query: 104 RSFLSQKGGSSDKRKMEEQRPKEHRPKANENKPV 3
           RSFLSQKGGSSDKRKMEEQ+PKE RPKANENKPV
Sbjct: 11  RSFLSQKGGSSDKRKMEEQKPKEQRPKANENKPV 44
```

Figure A.4: Best RazerBlastS alignment (experiment 3)

```
>gi|222616956|gb|EEE53088.1| hypothetical protein OsJ_35848 [Oryza
           sativa Japonica Group]
          Length = 96

 Score = 62.4 bits (150), Expect = 2e-08
 Identities = 31/35 (88%), Positives = 33/35 (94%), Gaps = 1/35 (2%)
 Frame = -2

Query: 104 RSFLSQKGG-SSDKRKMEEQRPKEHRPKANENKPV 3
           RSFLSQKGG SSDKRKMEEQ+PKE RPKA+ENKPV
Sbjct: 59  RSFLSQKGGASSDKRKMEEQKPKEQRPKASENKPV 93
```

Figure A.5: BLAST alignment (experiment 3)

```
>gi|222616956|gb|EEE53088.1| hypothetical protein OsJ_35848 [Oryza sativa Japonica Group]
          Length = 96

 Score = 52.0 bits (123), Expect = 7e-13
 Identities = 23/27 (85%), Positives = 25/27(92.6%)
 Frame = -2

Query: 80 GGSSDKRKMEEQRPKEHRPKANENKPV 3
          G SSDKRKMEEQ+PKE RPKA+ENKPV
Sbjct: 67 GASSDKRKMEEQKPKEQRPKASENKPV 93
```

Figure A.6: RazerBlastS alignment that differs from BLAST A.5 (experiment 3)

Although it might appear that way, the gap in itself is not responsible for the alignments difference. Other RazerBlastS alignments contain gaps:

```
>gi|116778854|gb|ABK21027.1| unknown [Picea sitchensis]
           >gi|116788685|gb|ABK24963.1| unknown [Picea sitchensis]
           >gi|224284026|gb|ACN39751.1| unknown [Picea sitchensis]
          Length = 48

 Score = 59.3 bits (142), Expect = 1e-07
 Identities = 28/35 (80%), Positives = 32/35 (91%), Gaps = 1/35 (2%)
 Frame = -2

Query: 104 RSFLSQKGGSS-DKRKMEEQRPKEHRPKANENKPV 3
           RSFLSQKGG++ DKRK EEQ+PKE RPKANENKP+
Sbjct: 11  RSFLSQKGGAAADKRKSEEQKPKEQRPKANENKPI 45
```

Figure A.7: BLAST alignment with gaps (experiment 3)

```
>gi|116778854|gb|ABK21027.1| unknown [Picea sitchensis] >gi|116788685|gb|ABK24963.1| unknown
     [Picea sitchensis] >gi|224284026|gb|ACN39751.1| unknown [Picea sitchensis]
          Length = 48

 Score = 59.7 bits (143), Expect = 2e-15
 Identities = 28/35 (80%), Positives = 32/35(91.4%), Gaps = 1/35 (2%)
 Frame = -2

Query: 104 RSFLSQKGG-SSDKRKMEEQRPKEHRPKANENKPV 3
           RSFLSQKGG ++DKRK EEQ+PKE RPKANENKP+
Sbjct: 11  RSFLSQKGGAAADKRKSEEQKPKEQRPKANENKPI 45
```

Figure A.8: RazerBlastS alignment with gaps (experiment 3)

B MEGAN taxonomic trees

B.1 Gapped Alignments

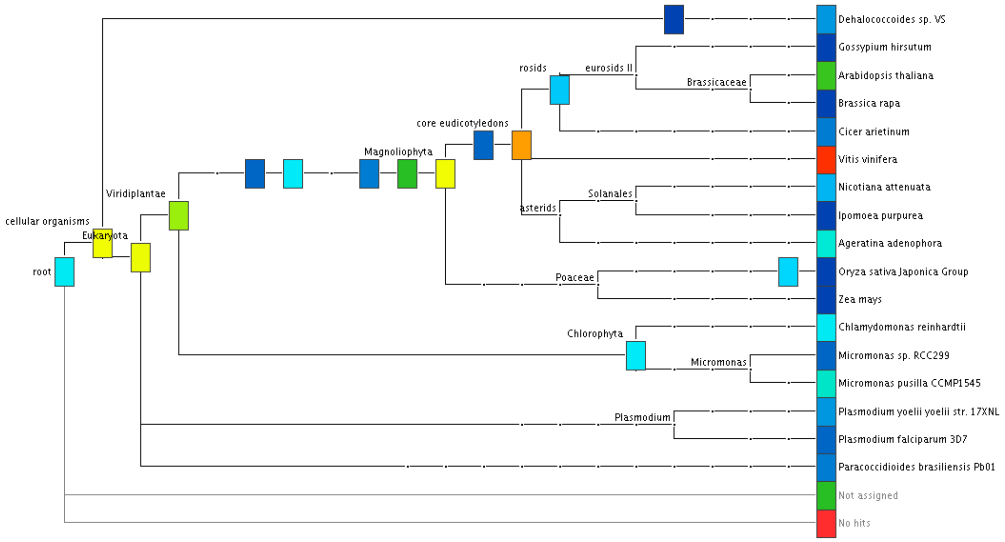

Figure B.1: Taxonomic Tree from gapped BLAST

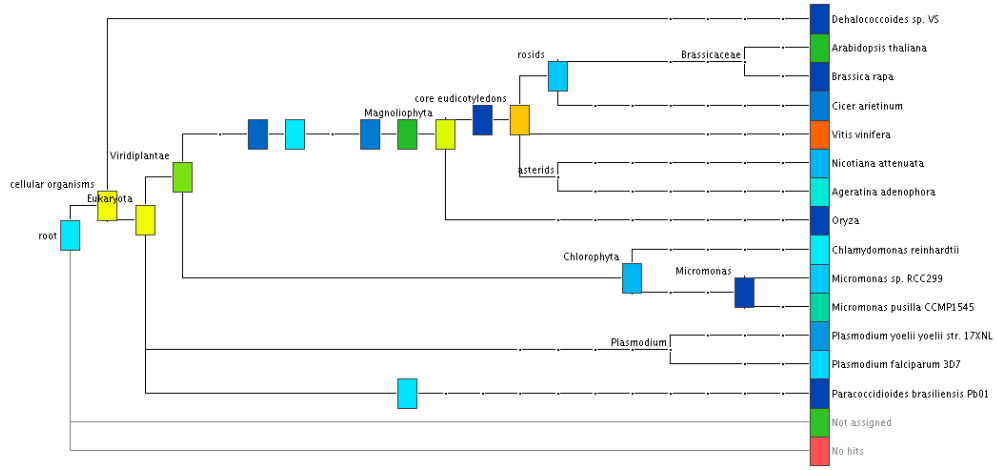

Figure B.2: Taxonomic Tree from gapped RazerBlastS (shape = 1101, threshold = 4)

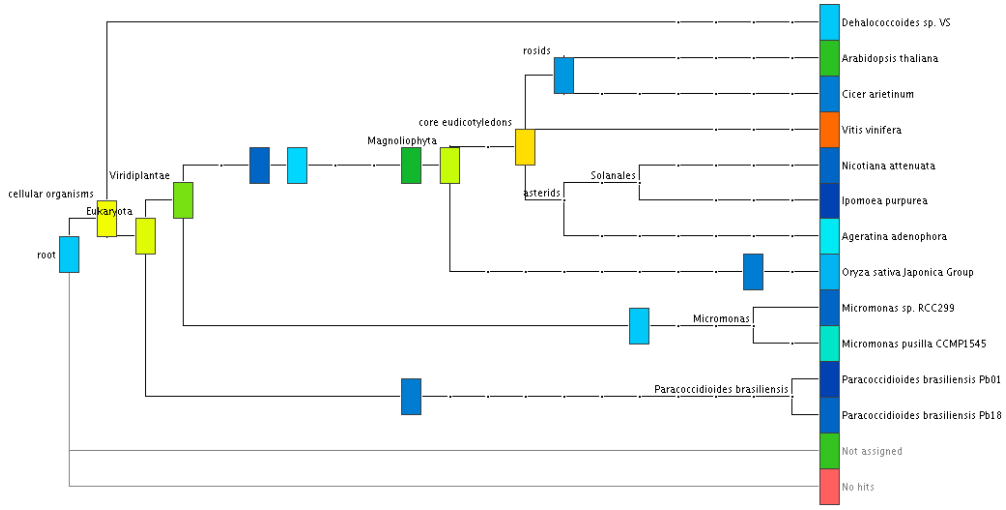

Figure B.3: Taxonomic Tree from gapped RazerBlastS (shape = 101101, threshold = 8)

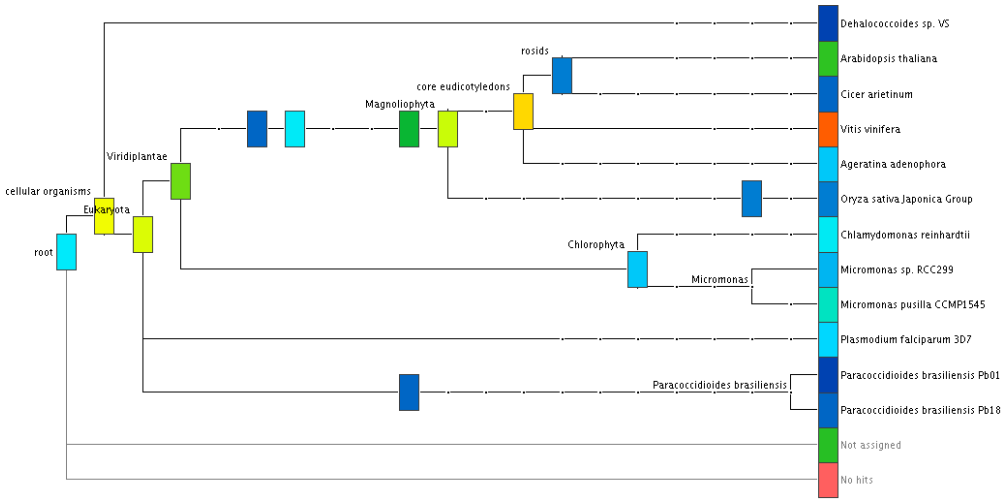

Figure B.4: Taxonomic Tree from gapped RazerBlastS (shape = 1011101, threshold = 3)

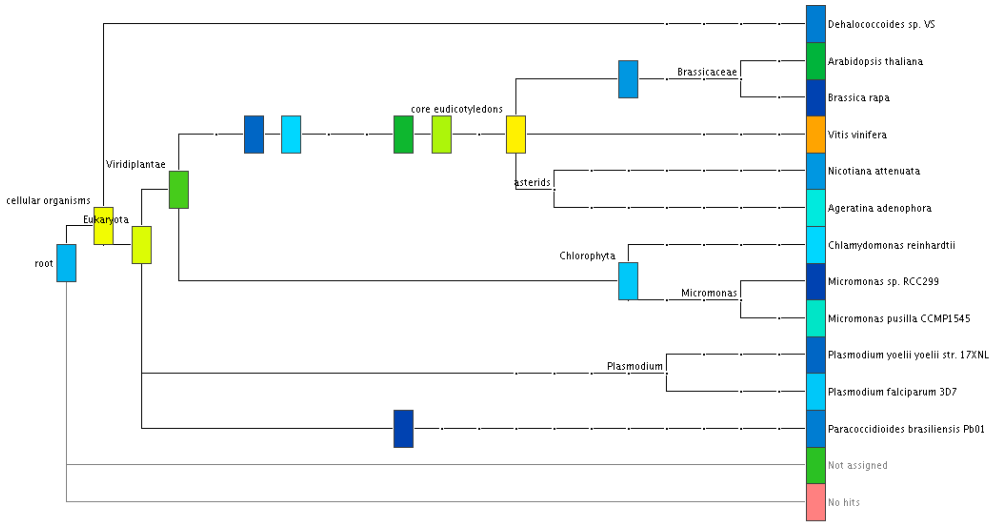

Figure B.5: Taxonomic Tree from gapped RazerBlastS (shape = 101101101, threshold = 1)

B.2 Ungapped Alignments

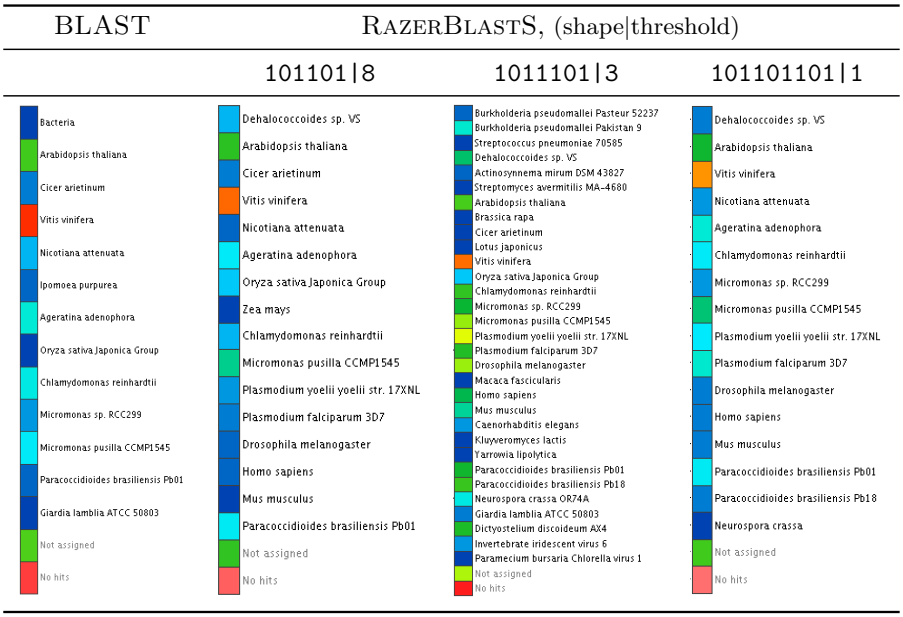

Figure B.6: Leaf nodes of taxonomic trees generated by MEGAN from ungapped alignments

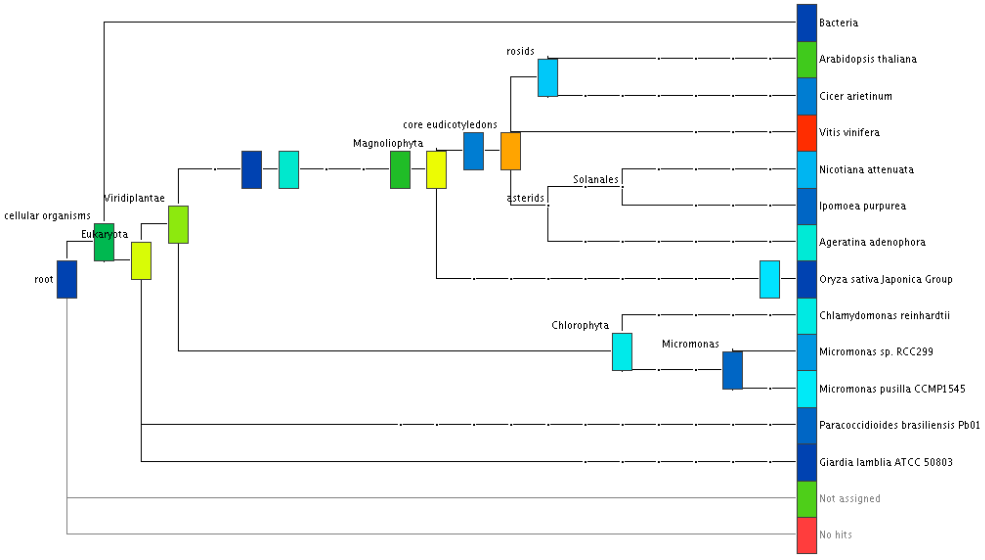

Figure B.7: Taxonomic Tree from ungapped BLAST

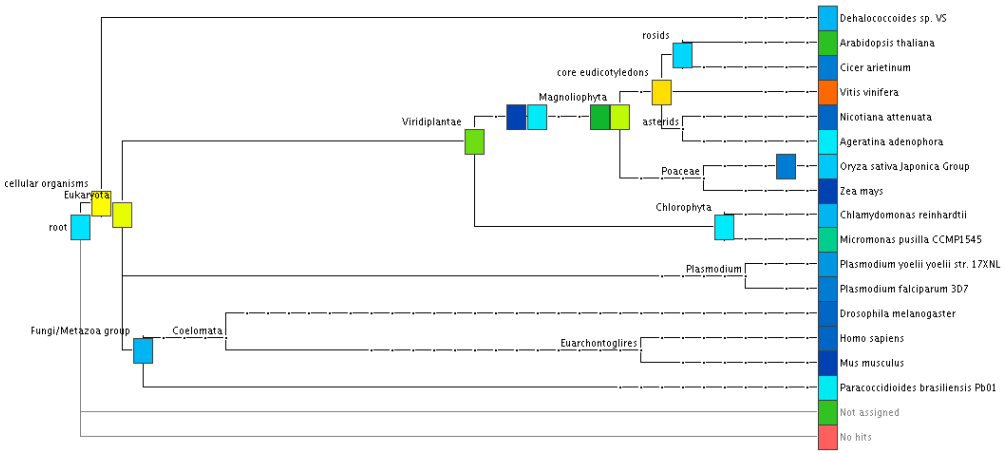

Figure B.8: Taxonomic Tree from ungapped RazerBlastS (shape = 101101, threshold = 8)

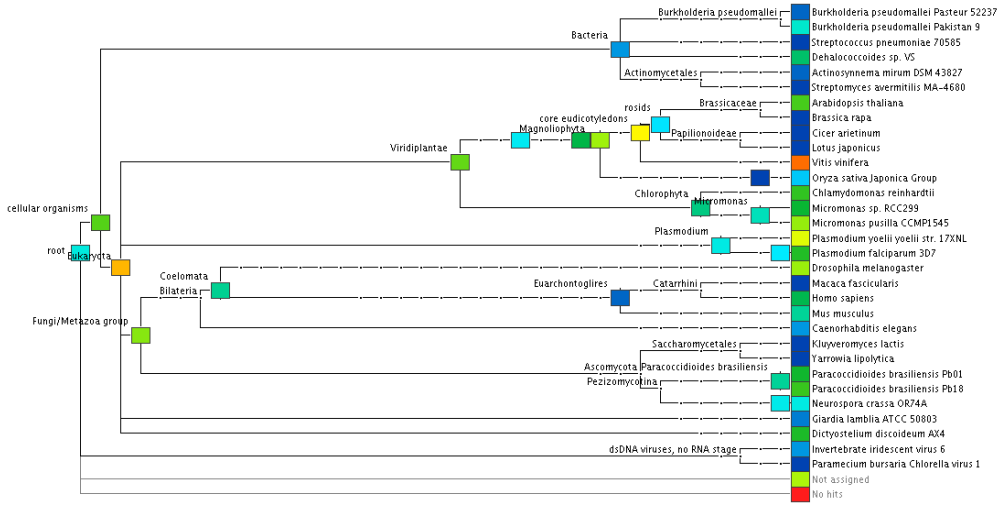

Figure B.9: Taxonomic Tree from ungapped RazerBlastS (shape = 1011101, threshold = 3)

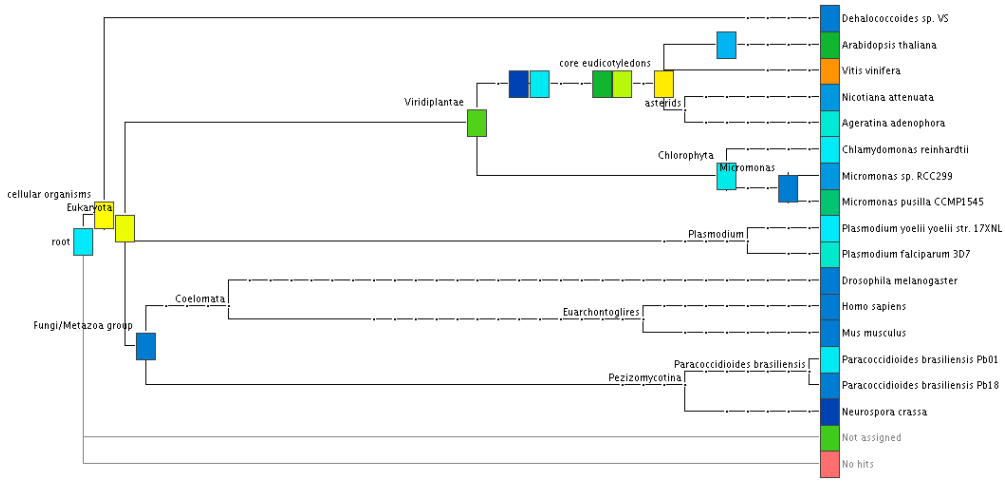

Figure B.10: Taxonomic Tree from ungapped RazerBlastS (shape = 101101101, threshold = 1)