Proceedings of the 2012 IEEE 12th International Conference on Bioinformatics
& Bioengineering (BIBE), Larnaca, Cyprus, 11-13 November 2012

Annotation Guided Local Similarity Search in Multiple Sequences and its Application to Mitochondrial Genomes

Ruby L. V. Moritz and Matthias Bernt and Martin Middendorf
Parallel Computing and Complex Systems Group
Institute of Computer Science, University Leipzig, Germany
Email:{rmoritz,bernt,middendorf}@informatik.uni-leipzig.de

***Abstract*—Given a set of nucleotide sequences and corresponding gene annotations which might contain a moderate number of errors we consider the problem to identify common substrings occurring in homologous genes and to identify putative errors in the given annotations. The problem is solved by identifying nodes in a suffix tree that contains all substrings occurring in the data set. Due to the large size of the targeted data set our approach employs a truncated version of suffix trees. The approach is successfully applied to the mitochondrial nucleotide sequences and the corresponding annotations available in RefSeq for more than 2000 metazoan species. We demonstrate that the approach finds appropriate subsequences despite of errors in the given annotations. Moreover, it identifies several hundred errors within the RefSeq annotations.**

I. Introduction

The past decades showed the importance of DNA sequence data in life sciences. New sequencing technologies decrease the costs and time to obtain new sequences vastly and thus increase the amount of available data [1]. In the light of the avalanche of new data efficient data analysis methods are indispensable. The quality of the available data is a prerequisite for all downstream analyses, e.g. sequence or gene order analyses.

The NCBI GenBank and RefSeq data bases provide genome sequences and their corresponding annotation [2], [3]. GenBank provides access to the data that has been submitted from labs all over the world through a common front end. Based on GenBank the aim of RefSeq is to provide a curated non redundant data set of higher quality. But due to the multitude of data sources and methods used for the annotation inconsistencies and errors are inevitable. The RefSeq annotations of metazoan mitochondrial genomes – which are the focus of this work – contain gene name inconsistencies and several cases of genes assigned to the wrong strand, missing in the annotation, or annotated superfluously [4]. Such problems cause trouble for automated large scale analyses and might lead to confusion for studies that focus on selected taxa.

To find homologous regions in two sequences the detection of very similar common substrings is often the first step, e.g. in the BLAST method [5]. The strictest kind of similarity is of course equality. The *suffix tree* data structure that stores the suffixes of sequences can detect such identical substrings efficiently [6]. Let S be a sequence over an alphabet Σ, $|S|$ denotes the length of S and S_i with $i \in \{1, \dots, |S|+1\}$ the suffix of S starting at position i. The suffix tree of a sequence S is a tree with $|S|+1$ leaves whose edges are labeled with substrings of $S \cup \{\$\}$, where $\$ \notin \Sigma$, such that the *path label* – being the concatenated labels of the edges along a path from the root to a specific node in this case leaf i – reads the suffix $S_i + \$$.

Suffix trees can be constructed in linear time and space using Ukkonen's algorithm [7]. A fundamental part of this algorithm are the *suffix links*. These point from a node with the path label αv with $\alpha \in \Sigma$ and $v \in \Sigma^{|v|}$ to the node with the path label v. Suffix links are also useful for a time efficient search in the tree. For an overview on suffix trees and applications see [8].

Generalized suffix trees determine common substrings in multiple sequences [9]. To achieve this all input strings $S_1, \dots S_n$ are concatenated like $S_1\$_1 \dots S_n\$_n$, where $\$_i \notin \Sigma$ and edge-labels are cut after any $\$_i$-symbol, for $1 \leq i \leq n$. If a leaf is allowed to contain suffixes of several sequences then a single separator $\$$ suffices. Generalized suffix trees are used in MUMmer for instance, that computes fast pairwise whole genome alignments using maximal unique matches [10]. For large amounts of sequence data the linear space required by suffix trees becomes problematic. One approach to deal with this problem are *k-truncated suffix trees* that restrict the length of the stored substrings to k [11], [12]. In [13] this idea was extended to multiple sequences, i.e. generalized k-truncated suffix trees. This data structure detects common substring up to a length k efficiently.

Note, that *suffix arrays* are more popular for storing suffix information then suffix trees [14]. But our approach benefits from the suffix tree structure. It might be possible to convey the method onto suffix arrays, but it would be less intuitive and comprehensible.

In this work we consider the following problem setting. Given are a set of nucleotide sequences and corresponding annotations, i.e. the start and end positions of the genes. Homology is assumed for genes with the same name. Furthermore the given annotations potentially contain a moderate number of erroneous annotations. Based on this data the goal of the proposed method is to determine common substrings that are part of homologous sequences. Therefor a generalized

k-truncated suffix tree is applied to detect potential indicators for genes. The proposed method may be used as an anchor or filtering method for future automatic *de-novo* annotation of DNA sequences. It is in particular promising for more expensive approaches, e.g. [15].

The result can be feed back to the given data to detect and correct contained erroneous annotations.

The over 2000 metazoan mitochondrial genomes that are contained in RefSeq are an ideal data set to evaluate our approach. The animal mt-genome is circular and usually comprises approximately 16k nucleotides [16] which typically encode 37 genes including 22 tRNAs, two rRNAs, and 13 protein coding genes. It is very dense compared to the nuclear genome. The control region, containing regulatory elements, is usually the only non-coding part. Mitochondrial genomes are often used to infer phylogenetic and evolutionary patterns among various taxonomic units [17]. They are also useful for studies in molecular ecology, biogeography, and their own molecular evolution that deviates vastly from that of the nuclear genome [18].

In the next section the new method is introduced. The results of its application to metazoan mitogenomes are presented in Section III. Conclusions are given in Section IV.

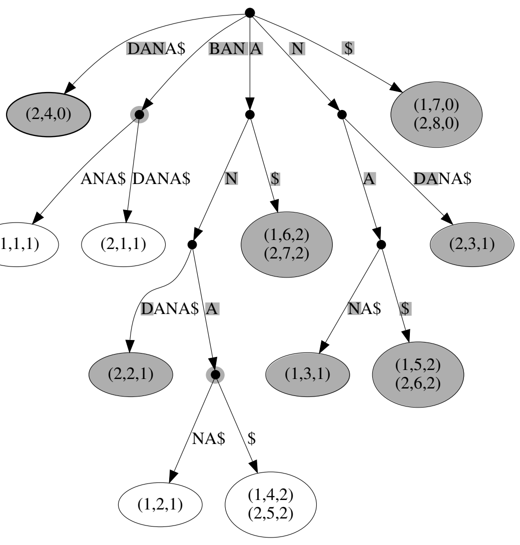


Figure 1: Suffix tree of S_1=BANANA and S_2=BANDANA, with $G = \{g_1 = \text{BAN}, g_2 = \text{ANA}\}$; the gray highlighted leaves (ellipses containing the indices) and edge labels are kept in the truncated version of the tree. Gray inner nodes are turned into leaves, that contain all indices of the leaves beneath them. The triplets in the leaves denote (sequence,index,gene).

II. Annotation Guided Suffix Trees

The method presented in the following finds similarities among multiple sequences by detecting substrings occurring in two or more sequences. To this end a generalized k-truncated suffix tree data structure is employed. This data structure is introduced in the following. Also, their use for detecting homologous regions and checking their usability as gene markers, and detecting tandem repeats are explained.

A. Usage of the k-truncated Suffix Tree

The input data consists of n sequences over an alphabet Σ. In this paper nucleotide sequences are considered and therefore $\Sigma = \{A, T, C, G, N\}$. Each character not in Σ that occurs in the sequence data is treated as N ($\approx$ 0.022%). For simplicity's sake N only matches with itself and e.g. $N - A$ is considered to be a mismatch. Furthermore, for each sequence an annotation providing the positions and strandedness of the genes that are indexed by an index set $G \subset \mathbb{N}_+$ is given.

Each inner node in a suffix tree corresponds to a string occurring at least twice as substring in the input sequences, i.e. the concatenated path labels occurring on the path from the root to the node. If a unique sequence end symbol is used per sequence the number of sequences where the string occurs is equal to the number of leaf nodes in the subtree rooted at the node. The positions of these strings occurrences can be computed, if the sequence and suffix index is known for each of the leaves. If the suffix tree is truncated to depth k then the leaves of the tree may correspond to more than one sequence and suffix index (the union of those from the truncated subtree). Thus the sequence index and the position of the suffixes are stored at the leaves [13]. Figure 1 shows how the suffix tree for two example strings changes due to truncation. Note that the truncation does not increase the asymptotic run time and the memory requirements are upper bounded by $\mathcal{O}(min(n, |\Sigma|^k))$.

For our application we also store information about the genes annotated at the corresponding positions at the leaves. A triple (s, p, g) — called *index* — denotes the fact that gene $g \in G$ is annotated at position p in sequence s ($g = 0$ is used for unannotated regions). This is the case if $g_0 \leq p \leq g_e$, where g_0 is the starting position of g and g_e its end position in s. Mind that p is the starting position of the word this leaf represents. Thus, every leaf has a set of indices corresponding to all occurrences of the substring it represents. Note that storing the indices at the leaves increases the memory requirements only by a constant factor since each position in the sequences is represented by exactly one index.

In [19] it was proposed to store the edge labels by their coordinates (starting position and length in input sequences), instead of storing them explicitly. In the annotation guided suffix tree, however, the edge labels are shorter (at most k) and storing them as characters is more efficient the more nodes the tree contains.

B. Annotation Guided Selection of Substrings

For a given index set one can ask if the corresponding sequences belong to homologous genes. In the absence of random hits and errors in the given annotations this would be reflected by indices pointing all to the same gene. Otherwise, some contamination in the index sets has to be expected. The strategy described in the following is employed to determine the correct gene in such cases.

A given index set I is partitioned into subsets $I_1, \ldots, I_{|G|}$ based on the gene index of the contained indices. The gene corresponding to the largest of these subsets (in case of ties it is chosen randomly) is denoted as the *target gene* of the index set I, i.e. the putative homologous gene sequence described by the index set. The reliability of this target gene is evaluated by its *precision* $P = (\max_{k \in [1:|G|]} |I_k|)/|I|$. Our approach is to accept an index set including more than one index only when its P value is larger than a given threshold. Accepting an index set with $P < 1$ can be interpreted as discarding some of the given annotations and replacing them with the target gene. The threshold adjusts the strength of evidence that is required to favor the designation of a subsequence to one specific gene over the candidate genes in the index set.

Storing the index set for each node of the suffix tree is not only prohibitive due to the required amount of memory but is also not necessary. Consider a leaf of the suffix tree and let L be the corresponding string that occurs on the path from the root to the leaf and let $I = (s, p, g)$ be an index attached to the leaf. Then, the index set of each node on the path toward the root contains I, because the corresponding prefix of L also appears at position p in sequence s and is included in gene g. Hence, the index set of a node can simply be computed as the union of the index sets of its child nodes.

The target genes and their precision values can be computed in a single postorder traversal of the suffix tree. In this traversal for each node the index set is computed as the union of the index set of the child nodes. From this each node's target gene and precision value can be determined. At any time at most k index sets need to be considered. This is because a node's index set can be deleted after all its child nodes were traversed and its target gene and precision value have been determined. Thus, only one index set for every node on the path from the root to the node currently traversed is stored. This path contains at most k nodes, as the tree is truncated at k.

Of interest are nodes that have a large index set on the one hand and correspond to long substrings on the other hand. Since the string length corresponding to a node increases but the size of the index set decreases with increasing distance to the root these two goals are conflicting and a good balance has to be found. This is done with the help of P. We choose at most one node on the path from any leaf to the root. In particular, the node that is the closest to the root satisfying the threshold is chosen and referenced to as *marker*. In a second traversal of the tree all markers are determined and stored along with their index sets. Any node below a marker can be ignored afterwards.

Note, that it is possible to further truncate the suffix tree by removing all subtrees below the markers. This would be useful if the suffix tree is then used to find the strings corresponding to the markers as substrings of a newly given sequence. Indeed it would suffice if the markers merely stored their target gene and P. The index information can be ommited.

To further truncate the tree, as proposed above, one can not simply prune the subtrees but has to take special care of the suffix links. Furthermore, the search procedure would need to be adapted to the situation that the suffix link of a node may now point to a node whose path label is shorter by more than one than its own path label. The following strategy can be used for the truncation. In a first iteration over the tree all nodes below the markers are tagged and their suffix links are redirected to the parent node of their marker via the tagged node's suffix link. In a second iteration for each node that is not tagged and has a suffix pointer that points to a tagged node the suffix pointer is redirected to the parent node of the corresponding marker. Finally, all tagged nodes are removed.

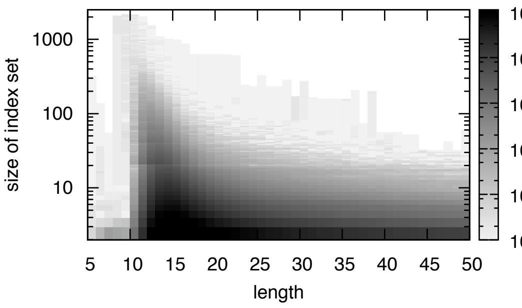


Figure 2: Histogram of the length of common substrings and the size of the index sets of the corresponding nodes selected as markers. The gray scale shows the amount of markers.

C. Altering Annotations

A marker with $P < 1$ contains indices pointing to *non-target genes*, i.e. all genes that are not the target gene of the marker. Each such case suggests to alter the annotation of a non-target gene in the respective species into the marker's target gene. We define the *score* S of such a putative alteration as the sum of the lengths of all markers that point to the annotated gene suggesting the same alteration. Any alteration with a score larger than the score of the annotated gene and larger than a predefined threshold is accepted.

To detect incorrect strand annotations the algorithm handles the complementary sequences of the genes as additional genes. This is, given the annotation of genes from a set G in the given sequences each gene $g \in G$ annotated on the complementary strand is stored as $-g$ in the indices attached to the suffix tree's leaves.

III. Results and Discussion

The behavior and properties of our method have been analyzed on the metazoan mitogenome sequences and their annotations given in RefSeq 45. This comprises the sequences and their reverse complements of 2149 mitochondrial genomes corresponding in total to 71×10^6 nt of sequence data. For mitochondrial genomes the algorithm differentiates between 76 potential target genes, i.e. the 37 mitochondrial genes and the control region in the two orientations. In the experiments the suffix tree is restricted to a depth of 50 and a threshold value $P = 0.95$ is chosen. The minimal score for the suggested alterations is set to 150. All computations were executed on a computer with $32\,\mathrm{GB}$ RAM and AMD Opteron™Processor 2435 with $2.6\,\mathrm{GHz}$.

The 50-truncated suffix tree data structure proves to be very efficient for our objective. Restricting the depth to $50\,\mathrm{makes}$ it possible to store the suffix tree in approximately $23\,\mathrm{GB}$ RAM. It includes 110.9×10^6 inner nodes and the path labels comprise 2.1×10^9 characters. More than half of the nodes is contained in subtrees below the inner nodes selected as markers (57.0×10^6 of the nodes and 1.0×10^9 characters).

Constructing the 50-truncated suffix tree takes between three and four minutes. Calculating P for every inner node, selecting the markers from the tree, checking them for tandem repeats, and printing their indices takes about $15\,\mathrm{min}$, as the tree has to be traversed twice.

A. Distribution of Markers

More than 7.5×10^6 markers are determined in the truncated suffix tree. The frequency and the length of the common sequences identified in the mitogenomes corresponding to the nodes in the suffix tree that are classified as markers is shown in Figure 2. Apparently there are markers with length exceeding $50\,\mathrm{nt}$. Although increasing the maximum depth might be favorable, it was not done because of memory restrictions. Despite the fact that only exact matches are considered many of the identified sequences are found in a surprisingly large number of sequences. Some exceptional markers of length $8\,\mathrm{nt}$ to $11\,\mathrm{nt}$ are even found over 2000 times in the input data. In 24 cases these are associated with the larger ribosomal subunit, in the remaining five cases they are associated with the smaller ribosomal subunit. Due to their near perfect conservation and a mean P of 0.98 each of these short sequences may serve as an indicator for the corresponding rRNA. The conservation in virtually all metazoan mitogenomes is surprising even though for mammalian mitogenomes the substitution rates have been reported much smaller for RNA coding genes than for synonymous sites in protein coding genes [20].

The distribution of markers in the different genes varies heavily (see Figure 3). But the differences are not obviously related to the gene type. The minimal average number of markers in protein coding genes is 0.42 for *atp8*. This value is similar to that of the tRNA coding gene *trnS1* with 0.46. Also the maximal average number of markers is similar, e.g. 0.82 for *cox1* and 0.75 for *trnM*. The values for the two rRNAs are medium compared to the values found for tRNAs and protein coding genes. It is interesting to contrast these results to data presented in [20], which propose that tRNAs might have more markers per nucleotide than protein coding genes because of their lower substitution rate.

Figure 3 also documents for how many sequences an annotated gene has no marker at all. The longer protein coding genes have generally more markers and are rarely missed in any sequence. Only the short *atp8* and *nad4L* are missed in some sequences. In comparison the tRNA-genes are not found more often. The maximum number of misses occurs for *trnS1*. One reason is that it is often shorter because it misses the D-arm (see [21] for a recent overview).

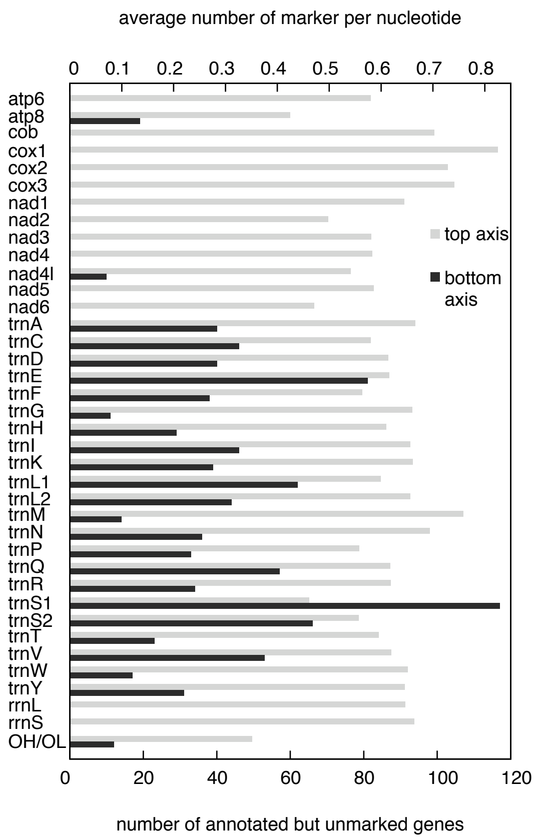


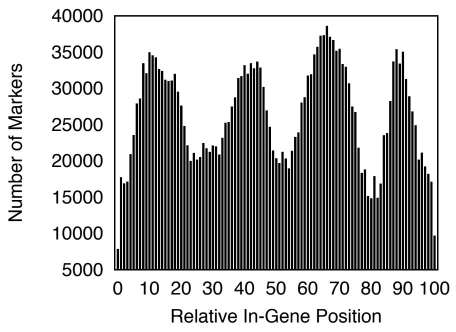
Figure 3: Portion of a gene that is covered by markers (light bars) and total number of sequences, whose annotation contains the respective gene, but has no markers in its position (dark bars).

Apart from the size and length of markers the position on their target gene is of interest. Although the tRNAs structure is of utmost importance for its function, the sequence itself can vary without changing the cloverleaf-structure. As Figure 4 shows there are regions in the tRNA that are far more often hit by a marker than others. In particular the middle of the gene, where the anticodon is, acquired many markers. This central peak appears in all other tRNA-genes and is not unique for *trnF*. The distribution of markers along the protein coding genes as can be seen for *nad1* in Figure 4, indicates that different regions of the protein structure show different degrees of conservation.

B. Annotation Errors

The vast majority of the markers (more than 7.3×10^6) hit their target genes with $P = 1$. The remaining 186 517 markers contain indices whose annotated gene deviates from the target gene (see Figure 5 top). The decreasing number of markers with P close to 1 can be explained by the fact that larger P require a larger index set which is less likely, e.g. for $P = 0.99$ at least 100 indices are required. The erratic number of markers for smaller P are caused by the fact that these values can be generated by different index set sizes.

Of the 161 990 annotated genes 76 565 contain marker



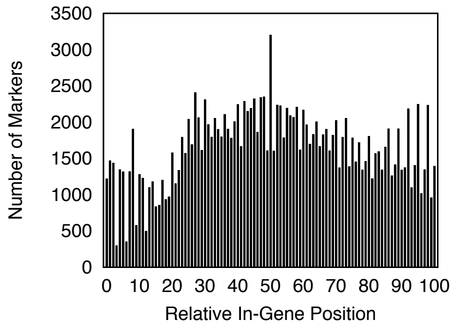
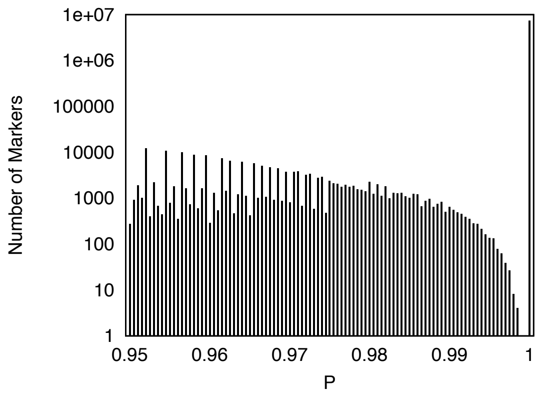


Figure 4: Occurrence of markers at different positions within *nad1* (top) and *trnF* (bottom); *nad1* has more markers than *trnF* as its sequence is longer.



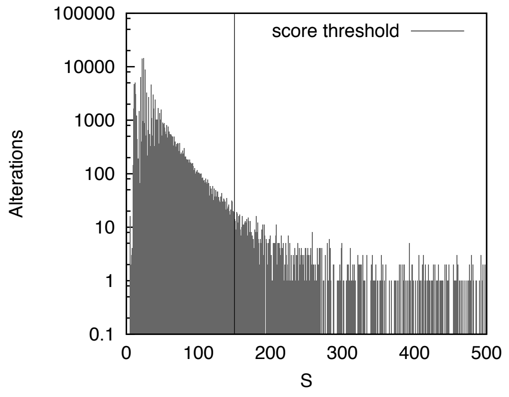


Figure 5: Histogram of P (top); Histogram of scores S of the suggested alterations (bottom), 436 alterations with $S > 500$ are omitted.

with a contradicting target gene. Often in such cases the target gene differs from the prediction of other markers or is only supported by a small number of markers containing few indices. Additional filtering is necessary to identify cases that give a strong indication for an erroneous annotation. This is done based on a threshold value for the score as defined in Section II-C. Table I lists all 16 cases with score ≥ 5000. Most of them are caused by obvious RefSeq errors, e.g. genes annotated over the whole genome. Figure 5 (bottom) shows the distribution of the score values. Out of the 210 733 suggested alterations 555 have $S \geq 150$. More than half of these can be classified to different errors as described in the following.

For both Leucine and Serine there are two tRNA-coding genes found in metazoan mitochondrial genomes, which are denoted as *trnS1*, *trnS2* and *trnL1*, *trnL2*, respectively. These tRNAs can be differentiated by their anticodons but are easily mixed up. The algorithm suggests to change the numerals of the *trnS* genes in 28 cases and in 49 cases for the *trnL* genes. In 5 cases the genes for the large and small ribosomal subunits are switched. Their different sizes helps to verify these errors. Although their gene name is not as similar as in the above cases, *trnE* and *trnQ* were exchanged in four cases. The amino acids associated to the corresponding gene products are glutamine (Gln, Q) and glutamic acid/glutamate (Glu, E) and where probably mixed up in these cases.

The strandedness is often erroneous in the RefSeq annotations [15]. The filtered markers propose to switch the strands of 203 annotated genes, 12 cases for rRNAs, 2 for protein coding genes and the remaining for tRNAs. Whether a gene is annotated on the reference strand or not depends

Table I: Annotation alterations suggested by the set of markers with $S \geq 5000$; the columns are accession number, annotation in RefSeq and suggested by markers (at the corresponding position), and a description of the cause ⋆: too long or too short RefSeq annotation; ◇: parsing problem; CDS: Coding Sequence (pseudo-genes miss a CDS)

Accession	RefSeq	Suffix-Tree		Problem in RefSeq
NC_001323	–	*OH*	◇	*OH* not annotated (parser: "D-loop")
NC_005044	–	*rrnS*+	⋆	387bp gap after *rrnS*
NC_006664	–	*rrnL*-		ribosomal genes missing (see also [22])
NC_006893	*rrnL*+	*cox1*+, *rrnS*+	⋆	*rrnL* annotated from 0 to 17202
NC_008131	–	*cox2*+	⋆	*cox2* annotated from 7206 to 7246
NC_008140	–	*nad1*+	◇	*nad1* missing (no CDS)
NC_008143	*trnH*+	*cox1*+	⋆	*trnH* annotated from 1166 to 11767
NC_008453	*trnY*-	*cox1*+	⋆	*trnY* annotated from 7 to 14586
NC_008649	*trnW*+	*cox1*+	⋆	*trnW* annotated from 5167 to 8098
NC_008683	*trnI*+	*rrnL*+, *rrnS*+	⋆	*trnI* annotated from 384 to 3923
NC_010292	–	*nad5*+	◇	missing in duplication (no CDS)
NC_010300	*trnM*+	*cox1*+	⋆	*trnM* annotated from 3915 to 13847
NC_012453	–	*rrnS*+	⋆	*rrnS* annotated from 1223 to 1244
NC_013635	–	*cob*+	◇	*cob* missing (no CDS)
NC_013810	–	*cox1*+	◇	*cox1* missing (parser conflict *cox3*)
NC_014679	–	*cob*+		3' end of *cob* in duplication

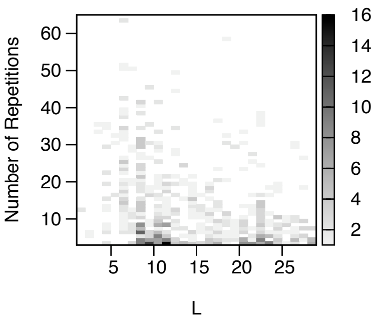


Figure 6: Number of detected repeats depending on the length L of their repeated unit and their number of repetitions.

on the inclusion of a "complement". Indeed in 173 cases the algorithm proposes to switch a gene annotated on the '+'-strand to the '-'-strand.

The mitochondrial genome is very dense and gaps in the annotation are thus suspicious, especially if the annotation misses genes. Our method proposes to fill these gaps in 76 cases, with either a missing or adjacent gene. Table I shows some of the more prominent cases. This concerns only protein coding genes, as the tRNAs are too short to get scores as high as 5000. Of the 184 remaining errors 95 can be shown to be related with one of the errors listed in Table I. The other errors include segments of length >100 that have competing gene annotations.

C. Repeats

Among the detected markers are repetitive structures. As tandem repeats have been reported in mitochondrial genomes in the control region [23], their efficient detection is of interest. Figure 6 shows the frequency and length of the detected repeated segments. The lengths of the repeated sequences range from 1 to 29 and they are repeated from 3 to 64 times. Of these repetitive segments 83 appear conserved in more than one sequence. The most extreme case is the 12 to 63 times repeat of `TACGCA` that is found in the control region of 16 different Mammalia.

IV. Conclusion and Outlook

A method was proposed that identifies common subsequences in homologous genes and can be used to correct erroneous gene annotations and to detect tandem repeats. The method was applied successfully to the metazoan mitochondrial genomes and has been used to identify errors in the corresponding RefSeq annotations. As it is currently designed mainly for mt-genomes, it is future work to adapt the method to other kinds of genomes. Especially the impact of large non-coding regions, introns, and duplications – phenomenons that are sparse in mt-genomes – has to be taken into account. As we work on biological data, future work should also consider the use of algorithms detecting similar instead of identical words.

References

[1] A. R. Jex, D. T. J. Littlewood, and R. B. Gasser, "Toward next-generation sequencing of mitochondrial genomes - focus on parasitic worms of animals and biotechnological implications," Biotechnol Adv, vol. 28, no. 1, pp. 151 – 159, 2010.

[2] D. A. Benson, I. Karsch-Mizrachi, D. J. Lipman, J. Ostell, B. A. Rapp, and D. L. Wheeler, "Genbank," Nucleic Acids Res, vol. 28, no. 1, pp. 15–18, 2000.

[3] K. D. Pruitt, T. Tatusova, and D. R. Maglott, "NCBI reference sequences (RefSeq): a curated non-redundant sequence database of genomes, transcripts and proteins," Nucleic Acids Res, vol. 35, no. Database issue, pp. D61–D65, 2007.

[4] J. L. Boore, "Requirements and standards for organelle genome databases," OMICS, vol. 10, no. 2, pp. 119–126, 2006.

[5] S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman, "Basic local alignment search tool," J Mol Biol, vol. 215, no. 3, pp. 403 – 410, 1990.

[6] P. Weiner, "Linear pattern matching algorithm," in 14th Annual IEEE Symposium on Switching and Automata Theory, 1973, pp. 1–11.

[7] E. Ukkonen, "On-line construction of suffix trees," Algorithmica, vol. 14, pp. 249–260, 1995.

[8] D. Gusfield, G. M. Landau, and B. Schieber, "An efficient algorithm for the all pairs suffix-prefix problem," Inform Process Lett, vol. 41, no. 4, pp. 181 – 185, 1992.

[9] P. Bieganski, J. Riedl, J. Cartis, and E. Retzel, "Generalized suffix trees for biological sequence data: applications and implementation," in International Conference on System Sciences, 1994, pp. 35–44.

[10] A. L. Delcher, S. Kasif, R. D. Fleischmann, J. Peterson, O. White, and S. L. Salzberg, "Alignment of whole genomes," Nucleic Acids Res, vol. 27, no. 11, pp. 2369–2376, 1999.

[11] J. C. Na, A. Apostolico, C. S. Iliopoulos, and K. Park, "Truncated suffix trees and their application to data compression," Theor Comput Sci, vol. 304, no. 1–3, pp. 87 – 101, 2003.

[12] J. Allali and M.-F. Sagot, "The at most k-deep factor tree," Institut Gaspard Monge, Université de Marne la Vallée., Tech. Rep., 2004.

[13] M. H. Schulz, S. Bauer, and P. N. Robinson, "The generalised k-truncated suffix tree for time-and space-efficient searches in multiple DNA or protein sequences," Int J Bioinformatics Res Appl, vol. 4, no. 1, pp. 81–95, 2008.

[14] M. I. Abouelhoda, S. Kurtz, and E. Ohlebusch, "Replacing suffix trees with enhanced suffix arrays," J. of Discrete Algorithms, vol. 2, no. 1, pp. 53–86, 2004.

[15] M. Bernt, A. Donath, F. Jühling, F. Externbrink, C. Florentz, G. Fritzsch, J. Pütz, M. Middendorf, and P. F. Stadler, "MITOS: Improved de novo metazoan mitochondrial genome annotation," Mol Phyl Evol, 2012, conditionally accepted.

[16] J. L. Boore and W. M. Brown, "Big trees from little genomes: mitochondrial gene order as a phylogenetic tool," Curr Opin Genet Dev, vol. 8, no. 6, pp. 668 – 674, 1998.

[17] C. Gissi, F. Iannelli, and G. Pesole, "Evolution of the mitochondrial genome of Metazoa as exemplified by comparison of congeneric species," Heredity, vol. 101, no. 4, pp. 301–320, 2008.

[18] J. W. O. Ballard and M. C. Whitlock, "The incomplete natural history of mitochondria," Mol Ecol, vol. 13, no. 4, pp. 729–744, 2004.

[19] D. Gusfield, Algorithms on strings, trees, and sequences: computer science and computational biology. New York, NY, USA: Cambridge University Press, 1997.

[20] G. Pesole, C. Gissi, A. De Chirico, and C. Saccone, "Nucleotide substitution rate of mammalian mitochondrial genomes," J Mol Evol, vol. 48, pp. 427–434, 1999.

[21] F. Jühling, J. Pütz, M. Bernt, A. Donath, M. Middendorf, C. Florentz, and P. F. Stadler, "Improved systematic tRNA gene annotation allows new insights into the evolution of mitochondrial tRNA structures and into the mechanisms of mitochondrial genome rearrangements," Nucleic Acids Res, vol. 40, no. 7, pp. 2833–2845, 2012.

[22] M. Perseke, G. Fritzsch, K. Ramsch, M. Bernt, D. Merkle, M. Middendorf, D. Bernhard, P. F. Stadler, and M. Schlegel, "Evolution of mitochondrial gene orders in echinoderms," Mol Phyl Evol, vol. 47, no. 2, pp. 855–864, 2008.

[23] J. V. Lopez, S. Cevario, and S. J. O'Brien, "Complete nucleotide sequences of the domestic cat (*Felis catus*) mitochondrial genome and a transposed mtDNA tandem repeat (numt) in the nuclear genome," Genomics, vol. 33, no. 2, pp. 229–246, 1996.