A semi-automatic methodology for localization of short mitochondrial genes in long sequences

Rafael Santos¹, José Humberto Machado Tambor¹, Luciana Campos Paulino¹, and Ana L. C. Bazzan²

¹ Laboratório de Genética Molecular e Genomas
Instituto de Pesquisa e Desenvolvimento
Universidade do Vale do Paraíba
Av. Shishima Hifumi 2911, Urbanova
São José dos Campos, São Paulo, Brazil 12244-000
{rafael, jtambor, lpaulino}@univap.br

² Instituto de Informática, UFRGS
Caixa Postal 15064
91501-970 – Porto Alegre, Brazil,
bazzan@inf.ufrgs.br

Abstract. Identification of short genes in long sequences using similarity measures (e-values and scores in BLAST queries) can be difficult in mitochondrial genomes since the similarity results of some genes can be shadowed by neighbor matches with higher similarity values. The same could happen for genes with relatively low similarity but which can be considered of interest in a particular study. In order to locate and identify those genes, a manual analysis of the similarity search results must be done, which can be time-consuming and error-prone.

In this report we present a methodology which aids researchers on the location of those genes by semi-automatically masking subsequences corresponding to genes that were already identified and limiting subsequent searches to the regions that did not present any result in previous steps.

A tool that implements this methodology was created and used in some database searches using a sequence obtained from a mitochondrial genome. We expected that analysis using this tool would be easier if not faster than the manual analysis.

Some results of the use of the tool are presented and compared with results obtained by manual similarity searching of BLAST results. As expected, the proposed tool didn’t present new results (i.e. different from the ones found in the manual analysis), since both rely on the same search mechanism, input and parameters, but the results were clearer in the sense of not being cluttered with similar results, and the shorter genes could be located more easily on the final similarity report.

Some comments on the classification of this tool as a software agent are also shown. Suggestions for improvements of the methodology and tool will also be presented.
1 Introduction

The mitochondrial genome is usually composed by adjacent genes with few base pairs separating the genes from each other. A good example is the mitochondrial genome of the yeast Saccharomyces cerevisiae [2]. Our laboratory is studying the mitochondrial genome of the fungus Paracoccidioides brasiliensis which size was estimated on being around 70kb using different restriction enzymes. Presently we’re on the final steps of sequencing by joining the contigs that were generated.

When one searches for similar genes using the BLAST (Basic Local Alignment Search Tool [1]) tool against the NCBI (National Center for Biotechnology Information) database, the genes which are more conserved will have higher similarity, and consequently the higher e-value and score values. When this happens, neighbor genes with lower similarity (or genes with relatively few base-pairs) does not appear on the comparison’s result (due to an user-chosen limitation on the number of hits to be displayed), or appear in positions where its relevancy cannot be adequately estimated.

Figure 1 shows the scheme of the Saccharomyces cerevisiae mitochondrial genome, illustrating this problem: when the complete genome is submitting to the NCBI database for comparison, the biggest regions possibly will be highly similar to sequences on the database, appearing in the beginning of the hit list. On the other hand, the smaller regions would not appear on the hit list or appear in a position so low (since the hits are ordered by e-values or scores) as to be considered as not relevant.

One obvious solution for this problem would be the manual analysis of the search result, considering as interesting hits (in the sense of possible similarity to a sequence of interest) the ones with high e-values and scores but with little overlap on already considered hits and the ones with acceptable e-values and scores in regions where nothing was previously considered as being interesting. In order for this to work, the query sent to NCBI should be formulated as to allow a large number of resulting hits and/or consider even hits with low e-value and scores, which can easily be done either via a
WWW form or via a command-line interface to the NCBI BLAST server. The main drawback of this approach would be the amount of work that the user would have to locate the non-overlapping hits on the search result.

This report presents a methodology that may help researchers that does this kind of analysis by semi-automatically selecting a subsequence that corresponds to a good hit on the database search and eliminating this subsequence from future searches, thus possibly eliminating overlapping hits and increasing the possibility of location of shorter genes that could be shadowed by large genes with higher similarity.

A tool that implements this methodology is also described. This tool could be considered agent-like in a broader sense – it could do a task on behalf of the user, taking some simple decisions to achieve the final results.

The next sections of this report presents some details on the methodology and tool that implements it, some examples of its utilization, conclusions and some considerations on the methodology which could lead to improvements.

2 A task-based approach for BLAST searches

The proposed methodology considers that in BLAST results with mitochondrial genome sequences as input gives as results several hits with relatively equal e-values and scores which are somehow related to each other and therefore redundant, while some shorter genes on the genome would appear in low positions on the hit list and therefore could possibly be ignored in spite of being important.

Ideally, to avoid this problem, one could subdivide the sequences in subsequences where 1) a gene or known region could be present or not and 2) there is little or no overlap with different genes of the same genome, ensuring that each subsequence would match a sequence in the NCBI’s database. Obviously this approach would require knowledge about the sequences that is not available before searching the database.

In order to solve this problem, we devised a simple algorithm that, when searching NCBI’s database, uses only a subsequence of the whole sequence of interest at a time, discarding regions on the whole sequence that were already matched in previous searches. Before presenting the algorithm, let’s define some terms that would be used through it.

A search task is a group of input sequence (with some ancillary information), execution script (that would perform the search in NCBI’s database using a BLAST client) and output results that is self-contained, i.e., it can be executed independently of other existing search tasks.

A task pool is a structure that is able to hold zero or more search tasks. The task pool can be implemented as a stack or queue, although it is possible to add or extract
more than one search tasks from it (for example, if multiple searches are to be executed in parallel).

The algorithm’s steps are as follows:

1. On the first step of the algorithm, create the (empty) task pool and the first search task. The first search task should use, as input, the whole sequence of interest $S$. The first search task should be put on the task pool.
2. Get a search task from the task pool.
3. Obtain a BLAST search result between the subsequence $s$ on the search task and NCBI’s databases using the search task’s script.
4. Store the result obtained in step 3. If no hits were found, no further processing should be done. If there were hits, parse the results in order to check which region of the subsequence $s$ corresponds to the best hit $b$ on the resulting hit list. Extract (or mask) from $s$ the region corresponding to the best hit $b$, creating as a result a number of new subsequences that can be zero (if the length of $b$ is equal or larger than the length of $s$), one (if $b$ is aligned left or right with $s$ leaving a remaining subsequence on the right or left side of $s$, respectively) or two (if $b$ is on the middle of $s$ with remaining subsequences on its left and right). Figure 2 shows graphically the possible subsequences that could be created from certain cases of $s$ and $b$.
5. If there were one or two subsequences in the previous step, create new search tasks with those subsequences and put them on the task pool.
6. Repeat from step 2 until the task pool is empty.

Fig. 2. Possible configurations for matching a sequence $b$ with a sequence $s$. On the top two lines $b$ is equal or larger than $s$, leaving no subsequences for further search. On the next two lines, $b$ covers the left part of $s$, leaving a right subsequence for further search. On the next two lines, $b$ covers the right part of $s$, leaving a left subsequence for further search. On the last line, $b$ cover a part of $s$ leaving subsequences on the left and right side for further searches.

The algorithm will stop naturally since there will be subsequences that will not create new search tasks, since the results won’t present any similarity hits, since it is common to pass to the BLAST client software some thresholds for e-values and scores, so it won’t return hits below those thresholds. Nevertheless, it is possible to further narrow the creation of new search tasks by 1) considering a minimum length for the subsequences $s$ and/or 2) avoiding creation of subtasks after some recursion depth.

Although the algorithm presented creates a list of search tasks (in contrast to the manual approach, that would require a single search against NCBI’s database), those search tasks would require just some few hits (or even just the best hit), and the results,
when combined, will be easier to understand.

A tool that implements the algorithm was created using the Perl language, which presented the following advantages over other considered options:

- Perl is tightly integrated with the Linux operating system we used, making possible the creation of the search tasks as new scripts that were put on an execution queue which is maintained by the system’s tools.
- There are tools for running BLAST clients and parsing its results written as modules in Perl (BioPerl [3]).
- Perl has parsing operators for other common tasks, specially for pattern matching and string processing, that proved useful for the implementation of the algorithm.

2.1 Some results obtained with the approach

To exemplify the presented methodology, a contig with 33936 base pairs (one of the largests found in the assemblies done in our laboratory) was used as input for the algorithm, and executed with the tool that implements it. For this particular analysis, we used the BLASTX program with a expectancy value of 0.00001 and showing only values which e-value was smaller than $10^{-5}$. The search was done against the NCBI database plus non-redundant proteins. A visual representation of the results can be seen on figure 3.

Figure 3 shows six levels of execution of the search tasks. The level 1 corresponds to a single search tasks using the whole contig as input, and the search task found the best hit on the database in the region between base pairs 22054 and 22752. This region was eliminated (not considered) for the next steps, and from the search task on the level 1 two new search tasks were created (one with a subsequence from base pair 0 to base pair 22053 and other from base pair 22753 to base pair 33935). Those subsequences

3 Created with a graphics editor – a visualization tool for the task pool is under construction.
were created as search tasks and included on the task pool, and the process repeated.

Some features of the algorithm can be seen on the visual representation of its results: from level 3 on, no hits above the specified thresholds were found, and no further search tasks were created from those branches. The same happened on levels 4 and 6, with short subsequences that didn’t generate any hits on the database search.

One interesting event happened in level 5: two hits were found for some short subsequences where the hit was larger than the subsequence (meaning that only part of the subsequence was enough to match a hit on the database). Those hits would be ignored since the regions were already matched in previous steps.

Figure 3 includes only search tasks up to the depth level six. For that particular contig the algorithm was able to create search tasks up to the level eight, but those were eliminated from the figure so it would fit in one page.

Since the search tasks were run over a few days’ period (due to server problems and Internet black-outs in our laboratory), we couldn’t gather reliable information on the time required to run the whole task pool and obtain the final results, but we consider a desirable feature of the methodology the ability to execute the search part by part over a period of time.

For a simple comparison with our methodology, we run a query with BLASTX considering a maximum of 1000 hits. A partial list of the 128 top hits is shown in figure 4. Although a complete comparison of the results with our approach and the manual analysis done with the results of a single BLASTX query could take some time to be done, our methodology’s results were considered satisfactory and in accordance with what was expected.

One can see in figure 3 that no hits were found on the region from around 25kbp onwards that region expresses tRNAs, therefore no hits were found in the databases we used for searches (using a different tool, 15 tRNAs were found between base pairs 25357 and 32034).

3 Conclusions

The methodology presented in this report was able to reduce the redundant information obtained with searches to the NCBI’s database using the BLASTX tool, and can be used to help on the localization of short genes (or genes which similarity values may be smaller than the ones for neighbor genes) on long sequences of mitochondrial genomes. The methodology could be used in a wider, more abrangent mitochondrial genome annotation system.

The software tool that implements the methodology can be considered as an agent if we use a less strict tipology (e.g. Nwana’s [5]), even if it doesn’t present some char-
characteristics that some authors consider important or essential for agents characterization.

One of the problems that genome annotation researchers face is that the best similarity (e-values and scores) obtained with BLAST searches to NCBI and other databases is often not enough to identify a sequence. Often, instead of considering that a sequence is similar to the best hit on the database search results, one researcher will choose another hit on that list because this hit has a description that fits better what was expected (e.g. one hit may have a smaller similarity value than another, but will be considered better since it is related to an organism closer to the one being studied).

Considering that problem and the fact that our approach gets always the best hit from the hit list, we consider our methodology as being semi-automatic since it does not annotate the mitochondrial genome by itself, and could, in some points, benefit from user interaction. Nevertheless, in the test runs we did with the tool, one of the authors concluded that he would make the same decisions as the tool, and the ones that would be done differently would impact little in the final result.

4 Directions of future work

Some improvements being considered on the methodology and tool are:

– The algorithm could be implemented in parallel, since more than one search could be performed at a time, and only search tasks that are ready to be executed are in the
We are considering the benefits of allowing its parallelization: since the longest subtask on the algorithm is the NCBI’s search (which is not being done in a local server), some tests would be necessary to verify whether the parallelization would result in a gain of execution time.

- We could solve partially the problem of selection of the best search result hit presented in section 3 by considering not only one best hit, but choosing, from each search task result, the $n$ best hits and allowing the user to choose one of those. The main drawback of this possibility is that each search task result may spawn a number of new search tasks between zero and $2n$, which could generate a further maximum number of $(2n)^2$ search tasks, and so on. Even if the number of search tasks would eventually cease to grow, it would be potentially very large in the beginning of the analysis. We are considering the benefits of using this approach, specially together with the ability of running searches in parallel.

- The integration of the methodology with a tool like Blast Search Update [4] could make easier to researches keep track of changes on the database searches and results. The use of other search mechanisms (such one used to locate tRNAs) could also be of value.

- The algorithm can have, during its execution, a pool of search tasks, and each one of those can be considered a simple agent. We could consider trying to implement the methodology using a multi-agent framework, but with agents being neither competitive nor collaborative.

5 Acknowledgments

Two of the authors are supported by FAPESP (Fundação de Amparo à Pesquisa no Estado de São Paulo) grants. Rafael Santos is supported by grant number 00/08705-3 and José Humberto Machado Tambor by grant number 01/09495-5.

The authors would like to thank professors Francisco Nóbrega and Marina Nóbrega of the Laboratório de Genética Molecular e Genomas of Universidade do Vale do Paraíba, for advice and assistance.

References