BIOINFORMATICS APPLICATIONS NOTE Vol. 24 no. 13 2008, pages 1530–1532
doi:10.1093/bioinformatics/btn223

Sequence analysis

PatMaN: rapid alignment of short sequences to large databases

Kay Prüfer*,†, Udo Stenzel†, Michael Dannemann, Richard E. Green, Michael Lachmann and Janet Kelso

Max-Planck Institute for Evolutionary Anthropology, Deutscher Platz 6, 04103 Leipzig, Germany

Received on March 21, 2008; revised on April 24, 2008; accepted on May 3, 2008

Advance Access publication May 8, 2008

Associate Editor: Limsoon Wong

ABSTRACT

Summary: We present a tool suited for searching for many short nucleotide sequences in large databases, allowing for a predefined number of gaps and mismatches. The commandline-driven program implements a non-deterministic automata matching algorithm on a keyword tree of the search strings. Both queries with and without ambiguity codes can be searched. Search time is short for perfect matches, and retrieval time rises exponentially with the number of edits allowed.

Availability: The C++ source code for PatMaN is distributed under the GNU General Public License and has been tested on the GNU/Linux operating system. It is available from http://bioinf.eva.mpg.de/patman.

Contact: pruefer@eva.mpg.de

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

1 INTRODUCTION

There is an increasing need to rapidly and accurately align short sequences to genomic or other biological sequences. Short sequence motifs, including restriction enzyme sites, microarray probe sequences, transcription factor binding motifs and miRNA sequences, are abundant in many areas of molecular biology. Identifying these short sequences is a crucial step in designing experiments and analyzing newly available genomic sequence data.

The most widely used approach for aligning sequences to large databases is the BLAST algorithm (Altschul *et al.*, 1990). Further optimized versions have been presented to speed searches for large numbers of sequences. The BLAST family of algorithms search for good alignments only where short, perfect seed matches between the query and target sequence exist. This heuristic vastly improves the overall speed by restricting the expensive alignment process to regions containing these short exact matches. There is a tradeoff between an extensive search and the speed performance of the algorithm. A search with longer seeds may miss some good alignments that contain mismatches or gaps, while shorter seeds will prolong alignment time. This tradeoff is especially severe for short query sequences because these may not contain a seed match to trigger full alignment, thereby missing good hits.

A well-known algorithm for searching multiple strings was introduced by Aho and Corasick in 1975. Although this approach has previously been implemented to search for restriction enzyme sites (Mount and Conrad, 1986; Smith, 1988), a comprehensive implementation for searches with mismatches and gaps is not available to our knowledge.

We developed PatMaN (Pattern Matching in Nucleotide databases), a tool for performing exhaustive searches to identify all occurences of a large number of short sequences within a genome-sized database. The program reads sequences in FastA format and reports all hits within the given edit-distance cutoff (i.e. total number of gaps and mismatches). We demonstrate the program's functionality by aligning Affymetrix HGU95-A microarray probes to the chimpanzee genome.

2 METHODS

2.1 Usage

The program accepts several parameters to specify a search. The user can specify both the maximum number of gaps and the total number of edits (gaps+mismatches) allowed in any reported match. Additionally the interpretation of ambiguity codes can be modified. When the ambiguity flag is set, any ambiguous character in the query sequences will be counted as a match if the aligning base is one of the nucleotides represented by the ambiguity code. When the flag is omitted, only the ambiguity code 'N' is allowed in the query sequences, and a base aligning to this character will be counted as a mismatch.

Both the query and target sequences must be in FastA format. The output is given in a tab-separated format containing the target and query sequence identifier, the start and end position of the alignment in the target sequence, the strand and the number of edits per match.

2.2 Algorithm

When initiated, the program begins constructing a single keyword tree of all the query sequences (Fig. 1). All bases along a query sequence are added as a path from the root of the tree to a leaf, with the edges representing the bases added, and the leaf node containing the query sequence identifier. If the user sets the ambiguity flag, all possible bases at ambiguous positions are added to the tree. If the user does not trigger the ambiguity flag, each base is added only once to the tree. The search for occurrences on forward and reverse strands is facilitated by also adding the reverse complement of all query sequences to the same tree. If an outgoing edge is not yet occupied after storing the query sequences, an additional suffix link is set to the longest existing suffix for the sequence represented by the path from the root to the node under consideration. The resulting graph will consist of internal nodes with outgoing edges for all four possible bases and for the ambiguity base 'N'. This procedure corresponds to the initial processing steps in the

*To whom correspondence should be addressed.

†The authors wish it to be known that, in their opinion, the first two authors should be regarded as joint First Authors.

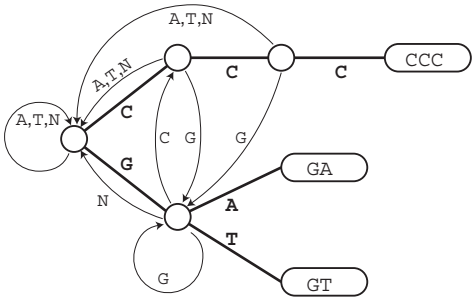

Fig. 1. Keyword tree with suffix links after adding the sequences 'CCC', 'GA' and 'GT'. The keyword tree (represented as bold lines) encodes the probe sequence as a path leading from the root node on the left side to the leaves on the right side. Suffix links are shown as arrows, but have been omitted at leaf nodes for brevity.

Aho–Corasick algorithm [for a complete discussion see Navarro and Raffinot (2002)]. Figure 1 depicts the resulting data structure for a small input example.

Once the tree is constructed, each sequence in the target database is evaluated base by base and compared to a list of partial matches. Each partial match consists of a node together with the number of mismatches and gaps accumulated. The list is initialized with one element containing the root node of the tree and an edit count of zero. In each iteration of the algorithm, all partial matches are advanced along a perfectly matching outgoing edge. Additional elements are stored for following mismatched edges and for producing all possible gaps, as long as the number of edits remains below the threshold given. If the outgoing edge is a suffix link, the resulting partial match is only included if no mismatch or gap occured in the part before the suffix. The number of edits needed to align the suffix is stored in the partial match when following a suffix link. Matches are reported when a partial match reaches a leaf node before exceeding the predefined number of allowed edits. The sequence identifier, match coordinates and number of edits are printed.

2.3 Complexity

When ambiguity codes are not interpreted and the query sequences contain no 'N' character, the keyword tree can be constructed in $\mathcal{O}(L)$ time and requires $\mathcal{O}(L)$ space, where L represents the total length of all query sequences (Navarro and Raffinot, 2002). When ambiguity is enabled, both time and space requirements increase exponentially in the number of ambiguity codes used in the patterns.

The time efficiency of the search algorithm is linear in the size of the target database, but depends heavily on the maximum edit distance as well as the average length and number of query sequences. For each additional edit operation, an exponentially increasing number of partial matches must be considered, since neighboring mismatched nodes and all possible gapped alignments are searched along with the perfect matching path through the tree. However, if only perfect matches are searched, the algorithm acts like the Aho–Corasick algorithm, and search time depends solely on the length of the target sequence. Time constraints therefore mean that PatMaN is only suitable for searching short sequences with a limited number of edit operations.

3 RESULTS

We used PatMaN to match 201 807 Affymetrix HGU95-A microarray 25mer probes to the chimpanzee genome (panTro2). The parameters chosen for this evaluation allowed up to one mismatch, but no gaps. The program spent ~2.5 h searching through all chimpanzee chromosomes and found 15.9 million hits (including 14.4 million hits to ALU repeat sequences). A table containing all unique hits to the chimpanzee genome is available on our website. Table 1 summarizes the time measured for conducting searches with different edit distance parameters using the same microarray probes and reads from one lane of the Solexa platform for chimpanzee chromosome 22. The measurement shows the exponential increase in runtime with the maximum allowed edit distance.

Table 1. HGU95-A probes and Bonobo Reads against Chromosome 22

Dataset	Edits	Gaps	Run time	Hits
HGU95-A probes[a]	0	0	0m13.31s	93 225
HGU95-A probes[a]	1	0	1m51.87s	327 028
HGU95-A probes[a]	1	1	3m36.92s	496 296
HGU95-A probes[a]	2	1	1h21m59s	1 843 008
Bonobo Solexa GAII data[b]	2	2	12h58m50s	14.3×10^9

[a]Benchmarking was performed on a 2.2 GHz workstation. Independently of the chosen parameters ~260 MB RAM were used. [b]Benchmarking was performed on a 1.8 GHz workstation and 8.6 GB of RAM was used during execution. The dataset contains 2.8 million reads of 38 bp length of genomic sequence from a Bonobo individual sequenced on the Solexa GAII platform.

4 CONCLUSION

We present a new tool for mapping short sequences to large nucleotide databases. The program does not require target or query database preprocessing and runs rapidly when a search is restricted to small edit distances. While we demonstrate the program's utility by aligning microarray probes, we anticipate further applications in the near future. In particular, mapping tags generated using next generation resequencing technology will require fast approximate matching to genomes to facilitate large-scale analysis of gene expression.

ACKNOWLEDGEMENTS

We would like to thank Christine Green for critically reading the article.

Funding: Funding has been provided by the the Max-Planck Society.

Conflict of Interest: none declared.

REFERENCES

Aho,A.V. and Corasick,M.J. (1975) Efficient string matching: an aid to bibliographic search. *Commun. ACM*, **18**, 333–340.

Altschul,S.F. *et al*. (1990) Basic local alignment search tool. *J. Mol. Bio.*, **215**, 403–410.

Mount,D.W. and Conrad,B. (1986) Improved programs for DNA and protein sequence analysis on the IBM personal computer and other standard computer systems. *Nucleic Acids Res.*, **14**, 443–454.

Navarro,G. and Raffinot,M. (2002) *Flexible Pattern Matching in Strings: Practical On-line Search Algorithms for Texts and Biological Sequences*. Cambridge University Press, New York, NY, USA.

Smith,R. (1988) A finite state machine algorithm for finding restriction sites and other pattern matching applications. *Comput. Appl. Biosci.*, **4**, 459–465.