Sequence analysis

MOODS: fast search for position weight matrix matches in DNA sequences

Janne Korhonen1,*, Petri Martinmäki1, Cinzia Pizzi2, Pasi Rastas1 and Esko Ukkonen1

1Department of Computer Science and Helsinki Institute for Information Technology, University of Helsinki, Helsinki, Finland and 2Department of Information Engineering, University of Padova, Padova, Italy

Received on July 14, 2009; revised on September 3, 2009; accepted on September 15, 2009

Advance Access publication September 22, 2009

Associate Editor: Alex Bateman

ABSTRACT

Summary: MOODS (MOtif Occurrence Detection Suite) is a software package for matching position weight matrices against DNA sequences. MOODS implements state-of-the-art online matching algorithms, achieving considerably faster scanning speed than with a simple brute-force search. MOODS is written in C++, with bindings for the popular BioPerl and Biopython toolkits. It can easily be adapted for different purposes and integrated into existing workflows. It can also be used as a C++ library.

Availability: The package with documentation and examples of usage is available at http://www.cs.helsinki.fi/group/pssmfind. The source code is also available under the terms of a GNU General Public License (GPL).

Contact: janne.k.korhonen@helsinki.fi

1 INTRODUCTION

Position weight matrices (PWMs), also known as position-specific scoring matrices or weighted patterns, are a simple, yet important model for signals in biological sequences (Stormo et al., 1982). For example, they are widely used to model transcription factor binding sites in the DNA. Due to the vast amount of biological data, both in PWM and DNA databases, high-performance algorithms for matrix search are needed.

Recent theoretical developments into PWM search algorithms can be roughly categorized into two groups, the index-based algorithms and the online algorithms. The index-based algorithms preprocess the target sequence into an index structure, typically a suffix tree or a suffix array, and use the index structure to facilitate quick search for matrix matches (Beckstette et al., 2006). The online algorithms, on the other hand, perform a simple sequential search over the target sequence. Most state-of-the-art algorithms of this type are based on classical string matching algorithms (Lefrere et al., 2009; Pizzi et al., 2007, 2009; Salmela and Tarhio, 2007; Wu et al., 2000).

While index-based algorithms may offer significantly faster search times, they also require a large amount of time and space for the construction of the index structure. For this reason, online algorithms are generally more practical in most situations, as typical DNA databases offer only raw sequence data. However, the work on advanced online algorithms has so far been mostly of theoretical nature, and no implementation packages intended for end-users have been published. To fill this gap, we have implemented a suite of efficient algorithms, called Motif Occurrence Detection Suite (MOODS). MOODS implements the algorithms developed in Pizzi et al. (2007, 2009), where also an extensive performance comparison of the new and old algorithms is reported. MOODS can be used as an extension to various scripting languages popular in bioinformatics. So far we have implemented bindings for the BioPerl (http://www.bioperl.org) and Biopython (http://www.biopython.org; Cock et al. 2009) toolkits.

2 ALGORITHMS AND IMPLEMENTATION

The core of MOODS is formed by the search algorithms themselves, implemented in C++ and making use of the C++ Standard Template Library. The package contains the following algorithms described and experimentally compared in detail in Pizzi et al. (2009):

- The lookahead filtration algorithm (LF) and its multi-matrix version (multi-matrix lookahead filtration algorithm (MLF)). For a given input PWM M, these algorithms first find the statistically most significant submatrix (i.e. the most selective submatrix against the background) of fixed length h, called the scanning window of M. Then the target DNA sequence is scanned with a finite state automaton that finds subsequences that score well against the scanning window. The full score against M is calculated only at these sequence positions. Scanning with the finite state automaton takes O(n) time, where n is the length of the DNA sequence, leading to nearly linear overall performance. The memory requirement of the finite state automaton is limited by the length h of the scanning window. In the multi-matrix variant, we combine all the automata into a single automaton, making it possible to efficiently find matches for a large PWM set in just one pass over the sequence.
- The naive super-alphabet algorithm (NS), which is as the naive matching algorithm, but uses a large alphabet consisting of tuples of original alphabet symbols. It works well for very long matrices (>30 bp).

The MLF algorithm is most suitable for PWM search tasks in practice and has the best overall performance of the algorithms of MOODS. For completeness, we have also included implementations of the naive algorithm, which directly evaluates the matrix score at all sequence positions, and the permuted lookahead algorithm.
With BioPerl and Biopython interfaces, the MOODS algorithms
are usable on any UNIX-like operating system supported by
gcc and either BioPerl or Biopython.

3 DISCUSSION

With BioPerl and Biopython interfaces, the MOODS algorithms
can easily be included into existing workflows. Likewise, scripts
can be written to use the implemented algorithms for specific purposes.
Existing facilities can be used to load sequences from formatted files or to fetch data from online databases. The results can then be processed further, for example, to find subsequences with statistically significant amounts of matches. On the other hand, the C++ algorithm implementations can also be directly integrated into existing or new software, thanks to the open source licensing. The MOODS web page (http://cs.helsinki.fi/group/pssmfind) provides several example scripts, as well as a simple C++ program for basic usage and as an example of C++ integration.

To benchmark the performance of our package, we tested the naive algorithm, the permuted lookahead algorithm and the MLF algorithm with real biological data. We did similar benchmark also for the Motility library (part of the Cartwheel bioinformatics toolkit; Brown et al. 2005), TFBS BioPerl extension (Lenhard and Wasserman, 2002) and Biopython’s built-in PWM matching algorithm. These packages all use the naive algorithm.

The test setup was as follows. We used matrices from the TRANSFAC public database (Matys et al., 2003) as our matrix set, containing a total of 398 matrices. The target sequences were taken from the human genome. We matched both the original matrices and their reverse complements against the sequences, in effect searching from the human genome. We matched both the original matrices and their reverse complements against the sequences, in effect searching for 796 matrices simultaneously. We ran the tests on a 3.16 GHz Intel Core 2 Duo desktop computer with 2 GB of main memory, running Linux operating system.

The results of our tests are displayed in Table 1. The results illustrate the advantages of carefully tuned C++ algorithm implementations and also indicate that more advanced algorithms offer practical benefits. We also tested matching the TRANSFAC matrices against both strands of the whole human genome with \( P \)-value \( 10^{-6} \), using the MLF algorithm. The total scanning time was about 42.1 min, with the number of matches being 29 354 584. Overall, these experiments indicate that our implementations perform well even on large datasets.

<table>
<thead>
<tr>
<th>Table 1. Algorithm benchmarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{600k} )</td>
</tr>
<tr>
<td>( P )-value</td>
</tr>
<tr>
<td>MOODS</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>TFBS</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Matches</td>
</tr>
</tbody>
</table>

We used two target sequences: \( \text{600k} \) is a 600 kb long human DNA fragment, and \( \text{Chr20} \) is the 62 Mb long human chromosome 20. The total scanning times for each algorithm or package are given, with ‘–’ indicating that the dataset was too large to be processed. The reported times include the construction of the data structures required in scanning as well as the scanning itself. The ‘matches’ row gives the total number of matches found for each \( P \)-value.

Funding: Academy of Finland (grant 7523004, Algorithmic Data Analysis), the European Union’s Sixth Framework Programme (contract LSHG-CT-2003-503265, BioSapiens Network of Excellence).

Conflict of interest: none declared.

REFERENCES