A simple Motif finder based on random projections

Author: Gergana Stanilova
Supervisor: Christopher Pockrandt

Department of Computer Science, Free University of Berlin, Takustr. 9, 14195 Berlin, Germany
E-mail:

ABSTRACT

In this report a motif discovery algorithm called *PROJECTION* is implemented in order to solve the "planted" motif problem more efficiently than other methods developed before it. It succeeds in finding the planted motif in a simulated dataset but with a huge runtime expense due to low computational power and lack of optimization of the implementation.

1 INTRODUCTION

Motif discovery methods are of great significance because they can point to meaningful biological conclusions. One application of motif discovery would be in the case of searching for transcription factor binding sites (TFBSs) in a genome. The TFBSs are sequences located upstream of genes to which a transcription factor binds and thus initiates the transcription of the genes in question. Therefore, if we want to influence the expression of a gene family we should look for a "motif" (a repeating sequence possibly with mutations upstream of every or almost every gene) that we can manipulate.

There have been numerous publications detailing motif discovery algorithms. One of the de novo methods, described by Pevzner and Sze (2000), is the so-called "planted motif" or (l, d)-motif search where l is the length of the motif and d the maximum number of mutations. While some of the solutions for this problem like Gibbs sampling by C. E. Lawrence and Wootton. (1993), and MEME by Bailey and Elkan (1995) had a poor performance for the (15,4)-motif problem, the one by Pevzner and Sze runs into difficulties for (14,4)-, (16,5)-, and (18,6)-motif problems.

In this report, I have described the implementation of a different algorithm to the motif discovery problem by Buhler and Tompa (2002) called *PROJECTION*, which solved the problematic cases mentioned above, by using random projections of the input sequences and an expectation-maximization (EM) algorithm to find a consensus sequence which corresponds to the motif.

2 METHODS

The *PROJECTION* algorithm was implemented in the language C++ with the help of the SEQAN-library developed by K. Reinert and Weese (2017). It consists of 3 STEPS: *Random Projections*, *Refinement* and *Consensus sequence*, repeated for m trials after which the best consensus sequence over all trials is chosen.

2.1 Step 1: Random Projections

Given t sequences of length n, we iterate through each sequence, create a projection of an l-mer onto a k-mer (k < l - d), hash the projection and group the l-mers, from which a certain projection originated along with the hash value into a bucket.

2.1.1 Projecting an l-mer onto a k-mer and hashing the projections

The projection of every l-mer in every sequence was done using a q-gram index from the SEQAN library. For the shape type, with which the q-gram index is specialized, *GenericShape* was used. *GenericShape* requires a bitmap (a string of chars) where a 0 represents a gap and a 1 represents the position of the character in the l-mer of which the k-mer consists. The bitmap was created by first adding k ones to a string of int-s, then adding l-k zeros and in the end, shuffling the string. Then with the SEQAN *hash* - function the hash values for each projection were computed.

2.1.2 Grouping the hashed projections into buckets

For grouping the hashed projections a map was created with the *hashValue* as the **map-key** and a vector of pairs as the **map-value**. The first element in each pair gives the number of the sequence, and the second - the position of the projected l-mer.

2.2 Step 2: Refinement

Knowing that the more elements a bucket has the more often the k-mer occurs, gives the advantage of being able to evaluate only some of the buckets - those with most k-mers. For that reason we define a threshold *s* for the minimal number of elements a bucket needs to have in order to be explored further. In the refinement step an EM-algorithm is used. Here the observed sequences is the known data and the positions at which the motif occurs are the missing data. First of all, an initial weight matrix is computed which later on gets used in the E-step of the algorithm to estimate a position matrix. Subsequently, in the M-step the weight matrix is optimized using the position matrix. These steps are applied until convergence.

2.2.1 Initializing the weight matrix

For every bucket h whose size is larger than the threshold s an initial weight matrix Wh is created and filled with zeros. Then Wh(i, j) is set to be the frequency of base i among the jth positions of all l-mers in h. Each time base i occurs in the sequences, Wh(i,j) gets incremented by 1 to get the absolute frequency. For the relative frequency, this number is divided by the number of elements in the bucket. Afterwards, a background probability is added in the end for the sake of avoiding a probability that equals 0.

2.2.2 Calculating the position matrix

In the position matrix *posM* the probabilities that the motif starts at a certain position in the sequence are stored. The goal is to estimate this missing information based on the initial weight matrix Wh with the following formula:

$$posM(i,j) = \frac{Pr(sequences|z_{ij}=1, W_h)}{\sum_{k=0}^{4} Pr(sequences|z_{ik}=1, W_h)}$$

where $z_{ij} = 1$ means that j is the starting position of the motif in sequence i. It was implemented by iterating through the sequences and for each l-mer's characters multiplying the corresponding probabilities from the Wh matrix. These values were saved as numerators, then summed for the denominators and in the end the numerators were divided by the denominators.

2.2.3 Iterating until convergence

The weight matrix was refined with the following formula:

$$W(0,0) = W_{A,0} = \frac{W'_{A,0}}{\sum_{i=A,C,G,T} W'_{i,0}}$$

for the first element of the matrix, where $W_{A,0}$ is normalized by dividing $W'_{A,0}$ (which is the probability that A is the 0th letter in the motif) through $\sum_{i=A,C,G,T} W'_{i,0}$ (the sum of the probabilities for every base being the 0th letter in the motif). In order to refine the weight matrix, the probabilities of the whole alphabet at the j-th position are updated and the process is repeated j times. To do that, we look at the sequence-matrix through a window, which contains all characters, that could possibly be on the current j-th position of the motif. The window has as many rows as the number of sequences, and extends for n - l + 1 columns. For each j in W the window is shifted to begin on the j-th column of the sequence-matrix and then the program iterates through all characters in the window to add their occurrence to the weight matrix.

2.3 Step 3: Consensus sequence

After the position matrix has converged, the estimated most probable positions of the motif can be used to extract an l-mer from each sequence. These l-mers are saved in a stringSet T from which a consensus sequence Ct is constructed by finding the most frequent charachter at every position among all l-mers. Moreover, a score for each bucket is calculated which equals to the number of hamming distances between each l-mer and the consensus sequence which is greater than d (the number of point subsitutions). For every bucket the score and its consensus sequence are compared and the pair with the lowest score is kept.

After these 3 steps had been repeated for every trial, again the bucket with the lowest score was chosen whose consensus sequence should correspond to the planted motif.

Table 1. Runtime of the different motif discovery algorithms

Motif finder	Runtime
PROJECTION	1h 30min
MEME	15.75 sec
Gibbs Motif Sampler	47.81 sec

2.4 Testing

For the purpose of validation, a dataset has been simulated with t=20 sequences of length n=600 with a planted motif *"AGGCATCCGTT"* of length l=11 with maximum d=2 mutations. This dataset was given as input in a fasta-file format in *PROJECTION* for m=16 trials, with a k-mer projection size k=7 and a bucket threshold s=4. The program was executed on Aspire VN7-571G-51WH using Linux Ubuntu 16.04 LTS.

The dataset was also uploaded to two online motif discovery tools: MEME from the MEME-Suite and Gibbs Motif Sampler (No. of different motifs = 1 and Motif Width = 11).

3 RESULTS AND DISCUSSION

All three algorithms managed to find the planted motif with 0 mismatches. An example with the same parameters but different data also had a high performance coefficient as described in Buhler and Tompa's paper. However, due to low computational power the runtime of *PROJECTION* was around 90 minutes, whereas for MEME and Gibbs Sampler it was within a matter of seconds as shown in table 1.

4 CONCLUSION

To summarize, the implemented algorithm had a successful outcome used on simulated data. Despite that, one test set is insufficient to prove the algorithm's efficiency and therefore more tests need to be run. Furthermore, using multiple cores for computing could significantly improve the runtime performance. Also, another aspect that could be considered is observing the cases in which the motif occurs zero or more than one times per sequence.

REFERENCES

Bailey, T. L. and Elkan, C. (Oct. 1995). Unsupervised learning of multiple motifs in biopolymers using expectation maximization.

Machine Learning, **21(1-2)**, 51–80.

Buhler, J. and Tompa, M. (2002). Finding motifs using random projections. *J Comput Biol.*, **9(2)**, 225–42.

C. E. Lawrence, S. F. Altschul, M. S. B. J. S. L. A. F. N. and Wootton., J. C. (8 October 1993). Detecting subtle sequence signals: a gibbs sampling strategy for multiple alignment. *Science*, **262**, 208–214.

K. Reinert, T. H. Dadi, M. E. H. H. S. M. R. R. J. K. C. P. J. W. E. S. G. U. and Weese, D. (2017). The seqan c++ template library for efficient sequence analysis: a resource for programmers. *Journal of biotechnology*, **261**, 157–168.

Pevzner, P. and Sze, S.-H. (2000). Combinatorial approaches to finding subtle signals in dna sequences. *In Proceedings of the Eighth International Conference on Intelligent Systems for Molecular Biology*, pages 269–278.