Getting the Topmost Scoring Sequences from Position Weight Matrices

Álvaro Abella Bascarán
alvaro.abella01@estudiant.upf.edu

November 27, 2014

1 Algorithm

1.1 Description

After a first glampse to the problem it came obvious that, as we must produce only the sequences with highest scores and in an ordered way, the first step would be obtaining the highest scoring sequence. Once we have this sequence we can use it as the seed to produce, by modification, new sequences in order of decreasing score. At this point our problem is subdivided in two subproblems: *a)* How to produce the highest scoring sequence? And *b)* How to obtain from it the n sequences with highest score? Here is the solution I found:

a) In order to produce the highest scoring sequence I first ordered by decreasing score the frequencies at each position (row) of the PWM. Once this is done we can just take the first nucleotide at each position and we will obtain the highest scoring sequence. Of course, there can be more than one sequence with the same score, but any of them will be an equally good starting point for our algorithm. However, in order to have more reproducible results, I chose a stable sorting algorithm (just using the pragma *use sort 'stable'*).

b) Producing from this sequence the n^{th} highest-scoring sequences was a considerably more difficult problem. The first step to take was fairly easy: if we have the top-scoring sequence (from now on, *sequence 1*), in order to obtain the second sequence we only have to modify the first one, applying the nucleotide change which decreases the score the least possible. After this we would have *sequence 1* and its modified version *sequence 2*.

   We now have a list of two sequences and ought to obtain the third one. We should iterate through each position of the two sequences and find the nucleotide change which produces the best possible scoring sequence different from the other two. We here encounter a problem, as we might find that the best change is the same as in the previous step, so that we would generate again the same sequence. To avoid this I decided to keep track of the changes applied to each sequence, such that if a given position of *sequence 1* was modified to obtain *sequence 2*, this position of *sequence 1* is marked now as *unmodifiable*, and will not be modified again.

   We have so far a working algorithm which starts with the highest scoring sequence and, at each step, produces the next top-scoring sequence, adding it to the list of top-scoring sequences. At each step all the sequences within this list will be analyzed to find the best nucleotide change, which will be applied to obtain the next sequence. There might be the case were there are several changes (say m changes) which produce equally scoring sequences, and in this situation all this changes are considered and applied to obtain m new sequences.

   During the course of the process it is probable that some sequence will be have been modified at all its positions, and thus should not be modified again. Before iterating over each position of each sequence, the algorithm checks whether this sequence is modifiable in at least one position and, if it is not, it just skips it.

   In order to produce only the n^{th} top-scoring sequences, we have to keep a count of how many sequences have been produced so far. Once the count has reached n, the main loop of the algorithm breaks and we return the list of all sequences, which are already ordered by score.

The command line interface to the program is provided by the file /src/main.pl. Below is the output of executing *./main.pl –help*:

```
Usage: main.pl [OPTIONS]... TRANSFAC_FILE
Produce the topmost scoring sequences from a position weight matrix.

  -n          number of sequences to be produced (by default, 10)
  -o          output file (by default, STDOUT)
  --log       use log-likelihoods as score (by default, absolute
              frequencies are used)
```

1.2 Efficiency of the algorithm

After an analysis of the algorithm and some mathematics, I have concluded that the time complexity of the algorithm is as follows:

Best case, with respect to l: $\mathcal{O}(l)$
Best case, with respect to n: $\mathcal{O}(n)$
Worst case, with respect to l: $\mathcal{O}(l)$
Worst case, with respect to n: $\mathcal{O}(n^2)$

2 Results

2.1 Motif 1

In the first case we have a very simple matrix, with zeros at many positions. We can easily see that the highest scoring position must be *GGACATGCCCGGGCATGTCC*, with only a tie in the last position, which can either be a *C* or a *T*. Running this program we should obtain that sequence as the first one (due to the stable sorting of each position), and the same sequence with *T* at the last position as the second one. Running the program to produce only the first ten sequences gives us the following results (in green, the nucleotides which don't vary across the ten sequences, in red, nucleotides which vary in at least two sequences):

./main.pl motif1.tf -n 10

GGACATGCCCGGGCATGTCC	307
GGACATGCCCGGGCATGTCT	307
GGACATGCCCGGGCATGTCG	302
GAACATGCCCGGGCATGTCC	300
GAACATGCCCGGGCATGTCT	300
GGACATGCCCGGGCATGTCA	300
AGACATGCCCGGGCATGTCC	298
GGACATGTCCGGGCATGTCC	298
GGACATGCCCGGGCATGTTC	298
AGACATGCCCGGGCATGTCT	298

The results conform to our expectations. There are two sequences with the highest possible score (which match perfectly the consensus letters of the *transfac* file), and we can see that the next sequences are mainly variations of the first ones in the last, second, and first positions. This is logical, as this positions are, in that order, the ones which show a smallest difference of frequency among the most frequent nucleotide and the second one. We can also see that the sequences are ordered by score. This is due to the algorithm, which produces only the top-scoring

sequences in an ordered fashion (the sequences are not sorted at any moment).

2.2 Motif 2

The results from the second motif are the following. In green, the nucleotides which don't vary. In red, those which vary in at least two sequences:

./main.pl motif2.tf -n 10

AATTTTCACGCATGAGTTCAC	591
AATTTTCACGCATGAATTCAC	590
AATCTTCACGCATGAGTTCAC	589
AATTTTCACGCTTGAGTTCAC	589
AATTTTCACGCATGAGTTCAT	588
AATCTTCACGCATGAATTCAC	588
AATTTTCACGCTTGAATTCAC	588
AATTTTCACGCATGAGTTAAC	587
AATTTTCACGCATGACTTCAC	587
AATTTTCACGCATGAATTCAT	587

Again the results are logical: our sequences are correctly ordered and each sequence is a variation in one single nucleotide of one of the previous sequences. The positions in red vary more due to the fact that the difference among the most frequent nucleotide and the second most frequent in that position is smaller, which makes it easier to change it maintaining a hight score.