Mining Episodes Satisfying Multiple Constraints.
Combinatorial Data Mining Algorithms (MTAT.03.249)
Seminar paper

Karl Blum
Timo Petmanson
25. december 2009

Abstract

In this paper we describe a method for doing sequence mining using multiple constraints, which is necessary when dealing with gene regulation problems. We describe a data structure called \textit{wild character tree} and an algorithm called \textit{DFSMiner} that could be used in data mining applications dealing with DNA regulatory areas.

Keywords: sequence mining, gene regulation

1 Introduction

Standard sequence mining treats the input as a single sequence and all locations of a pattern have same weight. In some gene regulation problems where the input consists of many different sequences (promoters) attached to different genes and some locations correspond more likely to a true biological sequence motif that binds proteins (transcription factors). This means that treating input as a single sequence is not useful when used on gene regulation problems. We have extended standard sequence mining techniques by adding this needed functionality. With the introduction of \textit{wild character trees} and \textit{DFSMiner algorithm} we are able to mine gene motifs with two different but parallel support types. In the following we discuss the techniques we implemented and in the end we make an experiment about how these techniques can be used on gene mining.

2 Safe over-approximation

The problem with support counting when treating each location in a pattern with an independent scoring value is that the definition of support are not downward closed any more. The sub-pattern of a frequent patter does not have to be frequent.

This is why we have to use safe over-approximation that is guaranteed to be downward closed [1]. The idea is that we have to fix maximum pattern length \(l\) and the we can calculate safe over-approximation value for a pattern with length is smaller or equal to \(l\).
3 Wild character tree

We have devised a data structure called wild character tree, that can be used for safe over-approximation type of support counting. This type of tree can be constructed from data in \( O(n \times m) \) time and getting support of any motif takes \( O(m) \) time, where \( n \) is the length of the data and \( m \) is maximum length of motifs we are mining. See Figure 1 for a high-level structure of a wild character tree.

![Figure 1: Example of a wild character tree.](image)

The labels of edges that form a path from root node to some intermediate node or leaf of the tree, define the motif whose support is saved in that given node. Also note that motifs like AT*** are represented as AT*. For example, consider a path:

\[
\text{root} \rightarrow A \rightarrow T \rightarrow * \rightarrow * \rightarrow \text{leaf}
\]

We know that if we have encountered an edge that represents a wild character, all following edges will represent wild characters as well as we use only fixed-length motifs for safe over-approximation. So we can just trim relevant intermediate nodes to save memory:

\[
\text{root} \rightarrow A \rightarrow T \rightarrow * \rightarrow \text{leaf}
\]

Such tree is very convenient for retrieving information from. Once the tree is built, it is trivial to find the path to the correct leaf containing support of some given motif. The complexity of information retrieval is \( O(m) \) where \( m \) is the length of the motif.

We will need one wild character tree per promoter as we need maximum scores of motifs of every promoter to calculate the maximal support.

3.1 Construction of wild character trees

The construction process of a wild character tree is very straightforward. We use the sliding window technique, thus making only one pass through the data, where on every iteration we add the motif with all it’s submotifs under the window to the tree with their relevant conservation and binding scores. See Figure 2 for an example.
We add all the submotifs into the tree in $O(m + 1)$ time, so the overall complexity of the process is $O((n - m) \times (m + 1))$ or simply $O(n \times m)$, where $n$ is the length of the data and $m$ is the sliding window size.

4 DFSMiner algorithm

To mine interesting motifs from a set of wild character trees we take advantage of the pattern growth principle and use depth-first traversal (DFS). Given a set of wild character trees we generate a set of initial candidates from all the available elements. We do not do any support counting and pruning in the initial stage because our algorithm is intended to be used on genome data where the number of different elements is small (4 nucleids \{A,C,G,T\}). This means that all possible elements become valid extensions or candidates for growing motif in each step.

Algorithm starts from an initial sequence and tries to extend it with possible extensions recursively. With each extension, the support of a motif is read from each wild character tree and the final support for a motif depends on the support type used (Total - the sum of supports from each tree is calculated; Average - the average support over trees). Both support types use different values from the wild character tree.

Using motif total support the pattern growth principle does not hold true, because the support is not downward closed, i.e. sub-pattern of a frequent pattern does not have to be frequent. Therefore covered support has to be used for pruning. After each extension we calculate the covered support from the wild character tree by extending the motif with an asterisk (*). It means ANY item and by using this we can get the maximum possible support for a given motif. If the covered support is less than given threshold then we can be certain that no possible extensions exist for this motif and this search path can be pruned. See Algorithm 1 for pseudocode.
Algorithm 1: MineMotif(motif)

if Support(motif) > threshold then
  save motif

foreach item do
  motif = motif + item;
  if motif length < n AND CoveredSupport(motif) > threshold then
    MineMotif(motif)

5 Experiment

We liked to see, how we could apply our data mining method to some real-world data. Mainly we wanted to see, whether using both conservation and binding scores in our data mining technique enables us to prune the search more, because if the binding and conservation scores are not correlated, we can prune using thresholds of both scores.

5.1 Data

We decided to use gene data of *Saccharomyces cerevisiae* (that is the most studied yeast so far). We collected the data from several internet databases [2] [3] [4] and we were able to compile conservation and binding scores for transcription factors of 98 genes. The exact pieces of data we gathered were:

1. Genes with their names, their nucleid sequences and locations in the chromosome. For example gene ECM1 contains 204 nucleids and is located from positions 36305 to 36509 in the chromosome:

   ECM1 location:36305-36509
   TGACAAAAATATATTGGGGCCCGCTCGGCTCATTTGTAGTATCTAAGATTATGTTATTT
   TCCTTTATATATTGTGTTATGAAACAGACAGAAGTAAGTTTCTGCGACTATATTATT
   TTTTTTTTCTTTTTTTTTTTCTTTATTTCAATTGCGATGAGCTGAAAATTTTTTTTG
   GTTAAGGACCCCTTTAGAAGTATTTG

2. Transcription factors of most of the genes. For example, gene NPR2 has two transcription factors: GAT1 and GCN4.

   ... NPP2: FKH1 NPR1: ABF1 NPR2: GAT1 GCN4 NRD1: RPN4 ...

3. Normalized conservation scores for nucleid positions in the first chromosome.
4. Normalized binding scores for some of the genes. For example, GAT1 has such binding scores for motifs of length 8.

\[
\begin{align*}
\text{TGAGTTAA TTACTCA} & \ 0.0716362813459118 \\
\text{TGATAAAA TTTTATCA} & \ 0.433933933933934 \\
\text{TGATAACA TGTATCA} & \ 0.481155095211698 \\
\text{TGATACAA TTGTATCA} & \ 0.265563815757833 \\
\text{TGATACCA TGGTATCA} & \ 0.356586000594034 \\
\end{align*}
\]

The process of putting the data together was very straightforward, for every gene’s transcription factor we directly mapped the conservation scores. As binding scores were given for every possible motif with length of 8, we simply traversed the nucleid sequence with a *sliding window* of size 8 and for every position in the sequence we took average of binding scores of motifs that matched given position in the transcription factor nucleid sequence.

5.2 Experiment

We show the results based on the experiment with only one gene, namely NPR2 and we only show process of mining with maximal support in this paper as results with additive support turned out to be statistically similar. Also the tests with other genes proved similar results.

The first step was to compare the distributions of the 100 most frequent motifs using only binding scores and using only conservation scores. The resulting histograms are given in Figure 3.

Figure 3: Histogram of 100 most frequent motifs with length of 7 using conservation score (left) and binding score (right)

![Histogram of 100 most frequent motifs](image)

We see that the distributions are not very similar as there seem to be more motifs with higher conservation score than with a high binding score.
Next we combined both scoring methods and mined 100 most frequent motifs with length of seven. We chose the thresholds based on the minimal scores of the frequent motifs we mined before and lowered them a little bit to get enough results. The conservation scores and binding scores of the motifs we mined using both scoring methods are given in Figure 4.

Figure 4: Correlation between conservation and binding scores for 100 most frequent motifs with length of 7

We see that the scores are not correlated at all. It even seems that all motifs with high conservation scores have rather low binding scores and vice versa. Thus, based on these results, it would make sense using both conservation and binding scores for pruning the search.

6 Conclusions

In this paper we described a method doing sequence mining using multiple constraints. We described safe over-approximation type of support along with wild character trees that makes this type of data mining applications possible.

The experiment we made, showed that using multiple constraints, it is possible to prune the search more efficiently than using only one constraint, given that the constraints are not correlated.
7 References

References


