Final Project

Report on

Spelling approximate repeated or common motifs using a suffix tree

Problem Description:

The proposed algorithm extracts repeated motifs from a sequence, typically of DNA, that is, a sequence defined over $\sum = \{A, C, G, T\}$. The motifs searched correspond to words over the same alphabet that occurs a minimum number q of times in the sequence with at most *e* mismatches each time.

The words representing the motifs may never be present exactly in the sequence. They therefore speak of the motifs, repeated in a sequence, as being "external" objects and denote them by the term *models*. They also call *q* the quorum constraint such a model has to verify to be considered valid.

This approach starts by building a suffix tree of the sequence and then after some further preprocessing, uses this tree to simply "spell" the valid models. It is therefore this tree that is now traversed to obtain such models.

In the following, *s* denotes (unique) sequence where repeated motifs are searched and by $\{s_i, 1 \leq i \leq N \text{ for some } N \geq 2\}$ a set of sequences from which common motifs extracted. In the case of DNA sequences, *s* and s_i are therefore elements of $\sum^+$ where $\sum = \{A, C, G, T\}$. A word in *s*, or s_i, called *u* if the sequence is equal to *xuy* with $x, y \in \sum^*$. The empty word is denoted by λ.

A model *m* is also an element of $\sum^+$. It is said to occur (or to be present) in a sequence *s* if there is at least one word *u* in *s* of same length as *m* and such that $Hamming(m, u) \leq e$ where $Hamming(m, u)$ is the Hamming distance between *m* and *u* (it is the minimum number of substitutions needed to transform *m* into *u*) and *e* is a non negative integer.

The repeated Motifs Problem. Given a sequence *s* and two integers $e \geq 0$ and $q \geq 2$, find all models *m* such that *m* is present at least *q* times in *s* (some of the occurrences of *m* may overlap).

Models satisfying the above conditions are called valid. We are therefore looking for all valid models that correspond to repeated motifs problem.

Constructing the Suffix Tree. Here are basic properties of the suffix tree *T* of a sequence *s*.

1. A branch of *T* may present any nonempty substring of s;
2. Each node of *T* that is not a leaf, except for the root, must have at least two offspring branches;
3. The strings represented by sibling branches of T must begin with different symbols of ∑.

The feature of a suffix tree is that for any leaf *i*, the concatenation of the labels of the branches on the path from the root to leaf *i* spells the suffix of *s* starting at position *i*. Indeed, in order to be able to spell the models present at least q times in s, all that remains for us to know is, for each node x of T, how many leaves are contained in the subtree of T having x as root. It is denoted by $Leave_x$ this number of each node x.

Method Description:

Spelling the Models

Valid models verify two properties when $e \geq 0$:

1. All their prefixes are also valid;
2. Spelling all the occurrences of a model m leads to nodes $x_1, ..., x_l$ in T for which $\sum_{j = 1}^{l}$ $Leaves(x_j)$ is at least q.

The algorithm is a development of the recurrence formula given in the lemma below where x denotes a node of the tree, *father*(x) its father and *err* the number of misspellings between the label of the path going from the root to x as against a model m.

Lemma 1. (x, *err*) is a node-occurrence of $m' = m\alpha$ with $\mathrm{m} \in \sum^{\mathrm{k}}$ and $\alpha \in \sum$ if, and only if, one of the following two conditions is verified:

(match) (*father*(x)*, err*) is a node-occurrence of m and the label of the branch from *father*(x) to x is α;

(subst.) (*father*(x)*, err* - 1) is a node-occurrence of m and the label of the branch from *father*(x) to x is $\beta \neq \alpha$.

A sketch of the procedure to follow is shown below for the case where models of a given length k are sought.

In order to do this model spelling operation, we have to make use of the following:

- a set Ext_m of symbols by which a model may be extended at the next step ;
- a set Occ_m of node-occurrences of a model m. We recall that these correspond in fact to classes of occurrences. Each node-occurrence x is represented by a pair (x, x_{err}) where x_{err} is the number of mismatches between m and the label of the path leading from the root to x in the tree;
- a variable *nbocc* that counts the number of "real" occurrences of the model we are currently trying to extend;
- a function *KeepModel*(m) that either stores all information concerning a valid model of the required length for printing later, or immediately prints this information.

The function *SpellModels* is called with arguments:

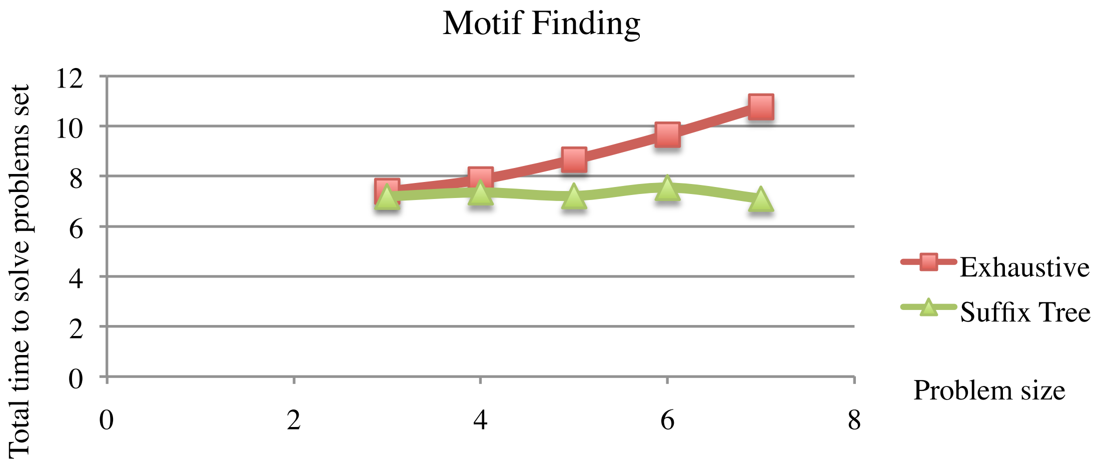
(0, λ, $Occ_\lambda = \{(root, 0)\}$, Ext_λ), where

Ext_λ equals to ∑ if $e > 0$ and otherwise, it equals to $label_b$ for branches b leaving the root.

$SpellModels(l, m, Occ_m, Ext_m)$

```
if ($l = k$)
    $KeepModel(m)$
else if ($l < k$)
    for each symbol $\alpha$ in $Ext_m$
        $nbocc = 0$
        $Ext_{m\alpha} = \emptyset$
        $Occ_{m\alpha} = \emptyset$
        for each pair ($x$, $x_{err}$) in $Occ_m$
            if there is a branch $b$ leaving node $x$ with a label starting with $\alpha$
                add to $Occ_{m\alpha}$ the pair ($x'$, $x_{err}$) where $x'$ is the node reached by
                following branch $b$ from $x$
                $nbocc = nbocc + Leaves_{x'}$
                $Ext_{m\alpha} = \begin{cases} Ext_{m\alpha} \cup label_{b'} \text{ for } b' \text{ leaving } x' & \text{if } x_{err} = e \\ \Sigma & \text{otherwise} \end{cases}$
            if $x_{err} < e$
                for each branch $b$ leaving $x$ except the one labeled by $\alpha$ if it exists
                    add to $Occ_{m\alpha}$ the pair ($x'$, $x_{err} + 1$) where $x'$ is the node reached
                    by following branch $b$ from $x$
                    $nbocc = nbocc + Leaves_{x'}$
                    $Ext_{m\alpha} = \begin{cases} Ext_{m\alpha} \cup label_{b'} \text{ for } b' \text{ leaving } x' & \text{if } x_{err} = e - 1 \\ \Sigma & \text{otherwise} \end{cases}$
      if $nbocc \geq q$
          $SpellModels(l + 1, m\alpha,_{m\alpha},Occ Ext_{m\alpha})$
```

Evaluation - log/linear Plot



Complexity

Assuming an alphabet of fixed size, the tree T can be constructed in $O(n)$ time where n is the length of sequence s and occupies $O(n)$ space.

Adding information to the nodes of the tree takes $O(n)$ and requires $O(1)$ space per node of T.

Concerning the spell operation, there are:

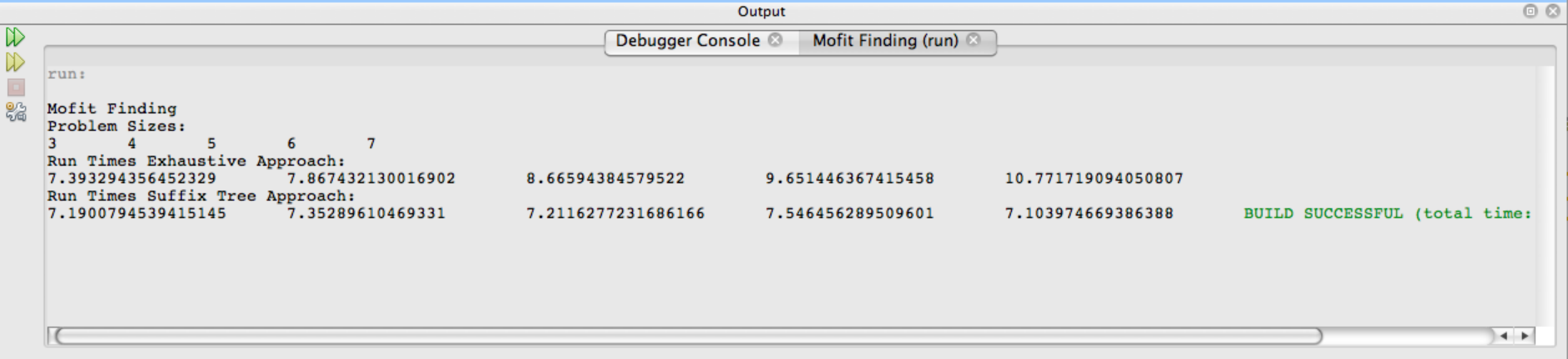
Lemma 2. Spelling all valid models for the Repeated Motifs Problem given T requires $O(n\, v(e, k))$ time where k is either the length of the models sought or is a maximum length.

Lemma 3. Spelling all valid models for the Repeated Motifs Problems given T requires $O(n)$ space.

Advantages and Disadvantages

There are some improvements on some pervious algorithms. It has also a better time bound whenever $N < k\,|\sum|$, and a better space bound when $N / w < k$.

A weak point is this algorithm doesn't know how to deal with gaps. This algorithm only find motif in a sequence of DNA instead of a set of sequences.



```
run:

Mofit Finding
Problem Sizes:
3       4       5       6       7
Run Times Exhaustive Approach:
7.393294356452329       7.867432130016902       8.66594384579522        9.651446367415458       10.771719094050807
Run Times Suffix Tree Approach:
7.1900794539415145      7.35289610469331        7.2116277231686166      7.546456289509601       7.103974669386388       BUILD SUCCESSFUL (total time:
```