Bachelor Thesis

Optimized pattern matching in genomic data

Report

Martin Westh Petersen - mqt967
Kasper Myrtue - vkl275

20. April 2015

Indholdsfortegnelse

1 Analysis

2 Loose_match

All possibilities are tried out using mismatches, insertions, deletions in that order to accept chars that don't match. When all mismatches, insertions and deletions are used, and we still encounter a mismatch, the stack is popped, to try a different order of the mismatches, insertions and deletions tried.

2.1 Scan For Matches

In order to understand what is good from Scan for matches we needed to understand some code segments and the overall structure of the program, simply put we came to understand this flow in the code:

$$\begin{tikzcd} & \begin{matrix}\text{B1}\\(m+1)\end{matrix} & \\ \begin{matrix}\text{A}\\(1)\end{matrix} & & \begin{matrix}\text{C}\\(2m-1)\end{matrix} \\ & \begin{matrix}\text{B2}\\(m+1)\end{matrix} & \end{tikzcd}$$

3 Design

3.1 Optimization

3.1.1 Skip length

At first the pattern is matched against the database sequence as usual with different orders of mismatches, insertions and deletions if necessary. This particular match (with a specific starting char in the DB-sequence SR) continues until the N chars from SR to SR + N in the DB-string are read, and the different letters counted, where N is the length of the pattern. The letters in the pattern are also counted. We now have 2 lists. One contains the different letters and number of occurrences in the pattern (patlist), and one contains the number of occurrences of letters of the DB-string in interval (SR, SR + len(pattern)) (datlist).

Say there is not match in this first run. Normally we would increment the SR counter and try again, but instead we increment SR, and update the datlist with the new letters. If the number of a specific letter in the pattern exceeds the number of allowed mismathces + insertions

3.2 Program structure

A list of actions our program should do in order to execute a pattern search:

- Read and parse the input line. E.g. ”scanFM ’ATTGCCCC[0,1,2]’ ’data.txt’”. Possibility of writing ”$->$ ’output.txt’” which results in the matches not being displayed in the terminal but written the the specified file.
- Parse the pattern into units and save them as different types (objects of different classes that inherit from a common PUNIT class), e.g. EXACT_PUNIT, AMBI_PUNIT etc.
- Choose the order of which to search for the patterns and create a state that readies for this search, for example a list of the punits in correct order with some way of keeping track of the different positions the punits have to with respect to each other.
- Search for the punits in the order chosen by simply calling a ”.search()” method for each PUNIT-object. The PUNIT-object’s search method invoked is unique for each different type of PUNIT, and returns either True or False. If the search for each PUNIT returns True the match is saved.
- The saved matches are either displayed in the terminal or written in a file, depending on the call of scanFM.

Types of PUNITs that we need

- EXACT - A PUNIT of this type consists of either of the letters ’A’, ’C’, ’G’ and ’T’ or any number of the wildcards.
-