Fast algorithm of finding seeds in a reference sequence

Ian Tsybulkin
Cloudozer
November, 2014

Introduction

This document describes the method of finding seeds or places where a query sequence may potentially match against a reference sequence. The approach suggests to generate a custom software code for a given query sequence that finds all possible patterns for a substring of a query sequence and then applies the generated software to find seeds. Then, an accurate alignment algorithm like Smith-Waterman local alignment can be used to sort out false-positive matches.
The probability of missing potential matches is also estimated.

Assumptions

We assume that a reference sequence is long, say 1 million bp or more. A query sequence is short, say 100bp. An acceptable match allows no more than a few defects (mutations or indels).

Approach

The described approach consists of three phases:
1. Generating custom code for seeds finding
2. Running the code to extract seeds
3. Running Smith-Waterman algorithm for the seeds found

Custom code generation

During this phase we generate a custom erlang function that takes a substring of our query sequence of some defined length and then generates all possible variations of this substring containing a defined number of defects (mutations or idels).

For example, a query sequence is “AAGTCAGTTAGCCA”. Thus, all possible variation of size 4 beginning from the very start of the query sequence are:

“_AGT”, “_AAG”, “AGTC”, “A_GT”, “A_AG”, “AGTC”, “AA_T”, “AA_G”, “AATC”, “AAG_”, “AAG_”, “AAGC”.

This phase takes less than a second and generates a function of a few hundreds of lines if the size of substring is lees than 10 symbols and we allow one or two defects in it. Longer strings having 3 defects or more could take a few seconds for compilation and consist of thousands of lines of code.

Seeds

The shorter the matching substring, the more seeds you will get, and thus, the more time you will need to spend for doing accurate local alignment later.

Long substrings find few seeds. However, they take more time for pattern generation and compilation.

For example, the substring of size 15, which contains 2 defects, found 3 seeds in a random reference sequence of size 1 million bp. It has about 1000 patterns and it takes less than a second for code generation, compilation and running.

Smith-Waterman vs. seeds finding trade-off

It takes approx. 6ms to align two sequences of size 100 agains each other on my MacBook Air laptop.

So, if we got 1000 seeds, it takes a few seconds to align them against a reference sequence.

If you have too many seeds, you may try to increase the length of a substring used for seeds finding. It obviously increases the time needed to generate, compile, and run the custom seeds finding code. When this time reaches the time required for SW algorithm you have an optimal balance between those parameters.

Can we miss a match?

We may easily estimate the likelihood of missing good alignments. Let us assume that our query sequence has length Lq. The substring used for seeds finding has length Ls. Na and Nd are acceptable number of defects and the number of defects we uses for seeds finding respectively. Thus the probability to miss a match, if it exists can be estimated:

$$p = ({}_{Na\,-}{}^{Na}{}_{Nd})(Ls/Lq)^{Nd+1}$$

if the number of mutations/indels in a query sequence is 3,
the number of mutations/indels that may happen in our substring is 2,
The length of a query sequence is 100,
and the length of a substring used for seeds finding is 10,
we get p = 0.1%

if Na = Nd we never miss any match.

Conclusion

It looks like the proposed method can work very well if we are trying to find an accurate alignments containing few mutations and/or indels.

If the number of mutations and/or indels is less than 5, the search of the seeds can be blazingly fast while the number of seeds is small, which allows us to obtain all alignments of our 100bp query sequence against entire human genome within a few seconds in a cluster of 100 commodity servers.