BLAT

Implementation in Python

Ómar Páll Axelsson

opa2@hi.is

We attempted to build an aligment tool based on BLAT where we try to identify homologous regions in the database by searching for perfect matches of K-mers, k = 11 that are relatively close together in the database. The data we used for testing was a subsequence of chromosome 12 as the database and query sequences from a transcripts file.

First we read the subsequence and the transcripts from the fasta files. We create an indexed list of nonoverlapping K-mers and their positions in the database. We do this by hashing each K-mer into a unique number. The K-mer is encoded into a base-4 number where A, C, G and T equal 0, 1, 2 and 3 respectively. This base-4 number is then converted to a decimal number which is used as an index in our indexed list. The position of the K-mer in the database is then appended to that specific index of our list. If a certain K-mer appears more than 10 times in the database, the positions of that K-mer are deleted and we flag it as a "common K-mer".

Then for each sequence in the transcripts we do the following: We set the query sequence as the transcript sequence. We build a "hit list" by hashing each overlapping K-mer of the query sequence and checking the index of our indexed list equal to the hashed number. If the index is empty there is no hit, else we create a new hit which is a list containing the following: the K-mer, the starting position of the K-mer in the query, the starting position of the K-mer in the database and the diagonal position which is the difference between the database position and the query position. The hit is then appended to the hit list. After the hit list has been created, the database is split into lists of size 64,000 which we call buckets. Each hit is then put into the appropriate bucket and after all hits have been put into a bucket, each bucket is sorted by the diagonal position of the hits using the built in sort() in python 3 which has a time complexity of O(N Log N).

Within each bucket we combine all hits sharing a diagonal position into a list called a "proto-clump". We do this by iterating through a bucket and checking the diagonal position of hit i against the diagonal position of hit i + 1. If they match, the hits are put into the same proto-clump. The hits are sorted by the diagonal position so no matches should be missed.

After the proto-clumps have been created we filter out hits that are further away than the window limit, W = 100, from another hit in the database. We iterate through each protoclump and compare the database position of hit i to the database position of hit i + 1. If the distance between the hits is less than W, we put them into the same clump. After all clumps have been made, if there are 300 bases or less between clumps they are merged into the same clump.

Now we build a homologous region from each clump by getting the database position of the first hit in the clump and the database position of the last hit. This is our range and we add 500 bases on each side to form a homologous region. We create a list of the ranges of these regions. From each range we create an indexed list as we did in the beginning but this time we don‘t check for common K-mers. We create a hit list between the query and each indexed list but this time if we get multiple hits for a K-mer we extend the K-mer until it‘s unique or k = 25. First we extend towards the left, that is we decrease the starting position of the K-mer in the query and the database. We check one letter at a time and continue until they don‘t match any more, then we switch over to the other direction. When we have a unique hit we put it in the hit list. When we have the hit list we merge hits that overlap, that is hits that have the same diagonal position and they share a substring. So we sort the hit list first by the database position of the hits and compare two hits at a time. If they overlap, then we merge them into one hit and leave the list containing the other hit empty. After all hits have been merged we delete the empty hits.

After the hits have been merged, we extend them into an alignment but without a limitation in regards to the size of the K-mer so that we get the largest possible K-mer that matches between the query and the database. After the extension, we splice together alignments that follow each other in the query sequence but are in different regions in the database. We give each alignment a score that is simply the amount of bases that match since we are only looking for perfect matches. We also specify the start and end both for query and database, the size of the query and identity which is 100% since we are only looking for perfect matches. We are only aligning to Chromosome 12 between 53 million and 55 million bases so we add 53 million to the database position. This is all kept in a list for each alignment which is printed out into the console.