Assignment 2

Bioinformatics Theory and Applications

Nitzan Shpigel 305388746

Elad Mizrahi 201481298

Part 2

discuss our BLAST implementation

Changing parameters affects:

Length of w-mer (W parameter):

W has great affect both running time and accuracy. From our experiments lowering W leads to a lower running time, lower sensitivity and higher coverage.
When W is lowered we can find neighbors faster, but probably we also have much more matches for every MSP, since the probability for finding string is higher, and it could affect the loss of sensitivity and accuracy.
Higher w will increase sensitivity, but we must be careful because for too high W we would not find any matches (or only few) and the accuracy may drop down sharply.
We noticed that for w=6 the running time was 9.5 minutes while with w=11 it was about 20 minutes. We can say that there is a linear connection between W length and the running time

Word threshold (T parameter):

T value effects are almost as W. Therefore, Higher threshold will produce more neighbors, which might affect loss of sensitivity, but higher T might filter too many words and the accuracy will drop down.
Luckily T has almost no effect on the running time, since it's only condition that determines whether add the neighbor or not, and this could be done in $O(1)$.

Extension threshold (X parameter):

The X parameter determines the score for each MSP, Therefore it affects the sensitivity.
High parameter value will lead to higher coverage (longer MSPs), which in turn may lead to false-positive score identification.
Lower value will raise the score and the sensitivity but also restrictive flexibility so we might miss some MSPs which as lower score but better match.

Local alignment results from part 1:

Query	Best 2 texts
Q1	t2, t10
Q2	t9, t10
Q3	t2, t10
Q4	t9, t10
Q5	t9, t10
Q6	t4, t10
Q7	t2, t10
Q8	t2, t4
Q9	t2, t10
Q10	t2, t4

Our blast implementation used three parameters: w, T, X.
It's easy to see that w and T are strongly connected each other, because the word's threshold depends on its length. For example: for a word in length w the maximal score can be obtained for the w-mer is $w \cdot 5$ and the minimal is $w \cdot (-4)$.
The X threshold depends the number of mismatches when extending the msp, and because mismatch worth -4 we decided the jumps should be in size 4 for each combination.

Using intuition, previous scores and the rules mentioned before we calculated some combination that should give us better results. Each cell represents the **percentage of matches** comparing to the local alignment algorithm results from part 1.
Notice that this isn't the whole table, and we filtered some other experiments.

w	6				9				11			
T	15	20	25	30	15	20	25	30	15	20	25	30
X = 0	50%	50%	30%	30%	65%	45%	45%	20%	**70%**	25%	25%	20%
X = 4	55%	55%	25%	25%	65%	35%	35%	15%	65%	35%	35%	20%
X = 8	55%	55%	30%	25%	65%	45%	45%	25%	65%	45%	45%	20%
X = 12	60%	60%	25%	25%	65%	50%	55%	25%	65%	50%	50%	20%
X = 16	60%	60%	30%	30%	65%	55%	55%	25%	65%	60%	60%	25%

The main conclusion is that accuracy usually increase as much as w is lower, T is lower, and X is higher. However, if we continue this tendency too far (for example 5, 10, 20) the algorithm gives the same matches (9 and 10) for all queries (false-positive).
Finally, our optimized parameters are: W = 11, T = 15, X = 0 with 70% match.

Heuristic improvements:

Our greatest improvement is in preprocessing the query (i.e. step 2 of the algorithm). Instead of traversing all possible permutations of a w-long word in the query and scoring each to check whether it above the threshold, we look for any w-mer from the query for all neighbors of w-long in the text and checks if the words are similar.
This way we skip checking words that don’t occur in the text, thus reducing the running time of the pre-process while we don't lower its accuracy.
If the text isn't given in advance we add the option to use the naïve search.

Running time comparisons:

We will make few comparison for different parameters and calculate the running time.
Our best score – 70% match was given by the values: W = 11, T = 15, X = 0.
The running time is as follows:

```
Query q1 against all texts ran for: 41.8229542459 sec
Query q10 against all texts ran for: 53.0812992617 sec
Query q2 against all texts ran for: 171.024572142 sec
Query q3 against all texts ran for: 119.541789342 sec
Query q4 against all texts ran for: 332.375728765 sec
Query q5 against all texts ran for: 153.761416825 sec
Query q6 against all texts ran for: 118.870799734 sec
Query q7 against all texts ran for: 82.5988948111 sec
Query q8 against all texts ran for: 72.6823783254 sec
Query q9 against all texts ran for: 59.7817310272 sec

Total running time of blast algorithm: 20.1 minutes
```

After few experiments, we found that we can make the tool run **half of the time** faster if we reduce accuracy in 20%. We achieve it with the values: W = 6, T = 20, X = 0.
The running time is as follows:

```
Query q1 against all texts ran for: 22.5530273731 sec
Query q10 against all texts ran for: 29.4277986028 sec
Query q2 against all texts ran for: 82.4843045917 sec
Query q3 against all texts ran for: 62.3998549119 sec
Query q4 against all texts ran for: 132.350050034 sec
Query q5 against all texts ran for: 74.7514412008 sec
Query q6 against all texts ran for: 60.516358335 sec
Query q7 against all texts ran for: 43.9898489637 sec
Query q8 against all texts ran for: 39.1174201374 sec
Query q9 against all texts ran for: 33.8015119954 sec

Total running time of blast algorithm: 9.7 minutes
```

Unfortunately, all our efforts are still slower than the local alignment implementation from part 1. It was little unexpectable since blast algorithm should be faster thanks to heuristics. Probably our implementation isn't good enough as their professional blast which was developed by greats programmers in years of development ☺.

Scoring the results:

After all the steps our biggest problem was to think how we can use the extended MSPs to rank the "query vs text" and determine which 2 text are the most suitable.
To do so we did the following: for every MSPs we compute the total length of all MSPs over the query and text (separately for each), so if part of the query/text is covered by multiple MSPs that part is summed only once. In other words: if two or more segments are overlapped – we merge them. This creates consecutive segments that can be connected easily by local alignment.
This method increases the accuracy by avoiding false-positives for largely repeating sequences.

The algorithm bottleneck:

The major bottleneck is in the query preprocessing. The need to traverse all possible similar w-mer words for every w-long word in the query is a time-consuming task and should take about $O(|Q - w| \cdot |\Sigma|^w)$ for all neighbors. In order to handle it we ease the computation time using heuristics which discussed in the heuristic section above.

Memory requirement:

The major memory requirement for the tool are the hash tables (Dicts) created in parts 1 and 2 (i.e. the w-mer windows from the texts and the w-mers from the query and theirs neighbors). The space complexity can be lowered by two possible ways:
1. Using suffix-tree. Since many of the w-mers appears more than once (depends on w size) we can use suffix tree to dense it to more compacted structure. The disadvantage is the bit complicated implementation.
2. Maintaining Lempel-Ziv tree (Lempel-Ziv is a lossless data compression algorithm studied in Data structures course).
The tree containing coding for the words, and in every leaf the appropriate table.
This way we can lowering the number of nodes in the tree (i.e. entries in the table) but the disadvantage is the increasing in the time complexity.

Can we disregard indels?

Well, of course the answer is yes. BLAST algorithm itself is an example of an indel disregarding tool, and we know that it is pretty accurate in finding closely related sequences.
Secondly, according to biological finding, the probability of an indel mutation is very low compare to replacement mutation.
Finally, our tool is very versatile and the 3 parameters can be changed in many combinations to optimize the result and raise accuracy according to the input. Any indels will decrease similarity, while replacement can be ignored elegantly.
And that's why the answer is yes.