Thomas Holland

Problem

The problem my project aims to solve is regarding degenerate primer design for a subset of sequences. When creating degenerate primers there is the issue of making copies of DNA from organisms that the researcher does not want to copy due to primers that are too degenerate. By too degenerate I mean that they attach to DNA strands that we aren’t interested in due to the primer attaching to DNA that we don’t need nor want to be copied. This problem occurs often as its common to have mixed samples when taking them from an environment other than in a lab.

The first step of this problem is first actually marking DNA that you want to make copies of with degenerate primers. This problem has been thoroughly explored with solutions such as CODEHOP[1], ICODEHOP[2], Hyden[3], DePiCt[4], Primer BLAST[5], and Primer3[6] which are used to find a degenerate primer that matches sets of DNA. There is no research currently available that has approached the problem of stopping matches with confounding DNA sets.

The problem that I am aiming to solve is locating primers that amplify a given set of sequences. I will then compare these primers with the other DNA sets we don’t want to attach to and if it attaches to any of them, I will log which places it attaches. After checking all of the DNA in the set that I don’t want to make copies of my program will check to see which change in the primer would allow it to match with as many DNA strands as we want copies of while not making copies of the excluded set. It will then make another primer that will match with those not covered using the original. This will create two primers in place of one, so we will lose degeneracy but gain more primers to stop unintended attachments.

We will use a primer of minimum length 18 as the optimal range for primers is between 18 and 25. The length of the segment between the two primers will be equal to or larger than 140 nucleic acids long.

My approach

I am going to first attempt to use CODEHOP, ICODEHOP, Hyden, DePiCt, Primer BLAST, and Primer3 as they all make optimal primers when given a set of DNA. Assuming that one of them will provide an optimal enough solution for my needs I will be working with a set of degenerate primers and a set of DNA that I don't want the primer to attach to. I will have the Primers checked against each strand to see if they attach anywhere using the fixed pattern variable texts methodology by shifting it over by the amount that we can be sure it will not match by using the Knuth-Morris-Pratt algorithm. After finding the set of DNA that matches the primers from the set, we don't want the primers to match to I will take that subset of the points that match with the primers and mark which changes will have the least most in terms of removing degeneracy. This means whichever nucleic acid causes the most attachments with the set we don't want to attach to will have its degeneracy removed and we will move to having two primers to work with instead. It will then check the set left and continue this until we are left with a set of degenerate primers that attach to none of the DNA in the set that we don't want copied or at least the minimum number.

If CODEHOP, ICODEHOP, Hyden, DePiCt, Primer BLAST, and Primer3 don't provide a viable solution then this creating of the optimal primer with the set of DNA we don't want to attach to will be done while finding an optimum degenerate primer

Progress Report

When attempting to use the software's available from the list of CODEHOP, ICODEHOP, Hyden, DePiCt, Primer BLAST, and Primer3 some provided usable solutions. Unfortunately based on what I have read in the papers available for them I have determined that their solutions are not optimal for our purposes. The ones that I could use all provided different solutions to the problem because of the different processes they used. Some of the variance may also have been due to the variety of settings available. I attempted to make the settings as standardized as possible, but many provided some settings that the others did not, so they were unable to be edited. For my program it currently provides no settings. It reads in the fasta file and divides the different segments. Currently there aren't any markers to identify which sequences are in the set of sequences that we want copies of and which are not. The program sets the first 10 as the set that we want copies of from the 9745 sequences in the file. It currently checks for primers in the first 10 by brute force comparing them against all others in the set and marks where there are matches across the sequences. It does not shift the sequence currently, so it does not account for the likely instance that they are not perfectly lined up. It takes what is selected as the best degenerate primer and attempts to attach it to all of the sequences that we do not want it to make copies of. All of the sequences it attaches to are stored in an array for use later.

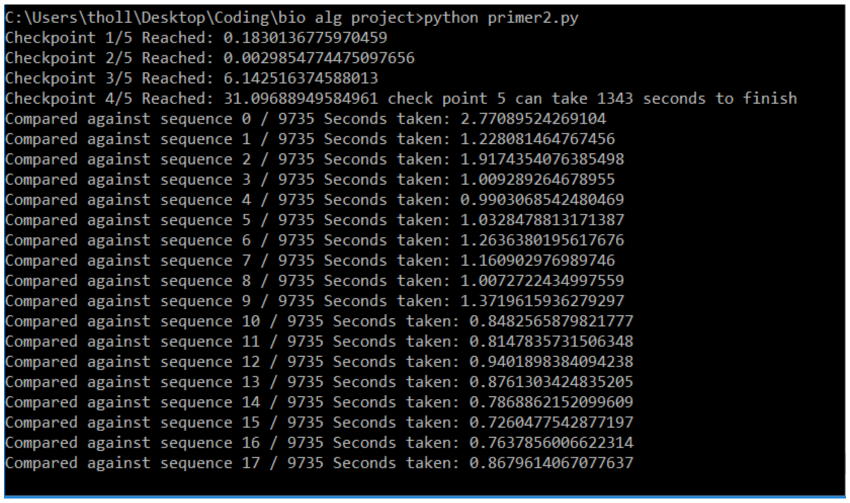
Future work

For the rest of the project the goal is to complete basic functionality of the program. This will be done by using the sequences that it attached to that it should not of and finding the areas that can have their degeneracy removed in order to not attach to as many of these unwanted sequences as possible. This will create more primers but will optimize for the problem of not wanting copies of these other sequences.

Final Report

Current program

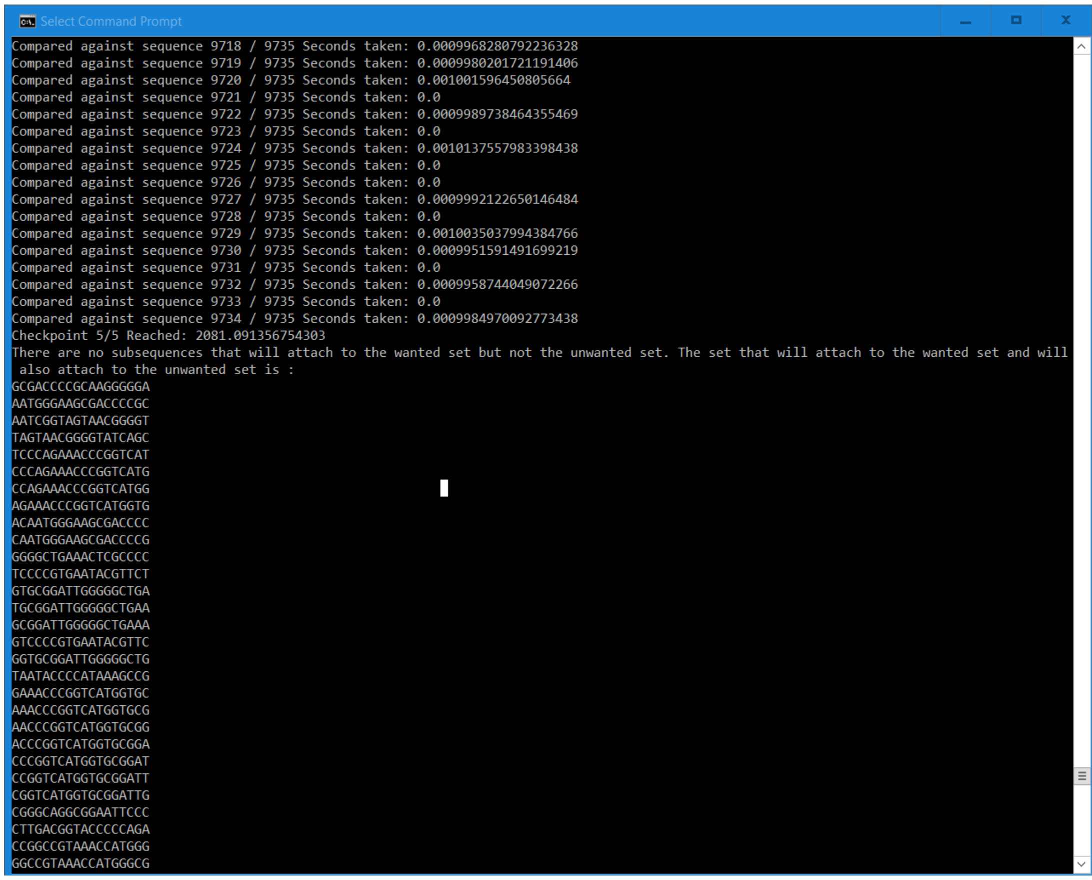
The program currently works by reading in the provided file and dividing it up into wanted and unwanted sequences. The wanted sequences are the first 10 in the file and are what we want to find primers for. The unwanted are the remaining 9735 sequences that were read in from the file that we want to prevent our primer from attaching to. The program then uses the first two wanted sequences to rank the subsequences in them. It does this by turning both sequences into lists of subsequences which are compared and based on the best match it finds across the two sequence lists it adds them to an appropriate ranking between 0 (no matches) and 18 (everything matches).



```
C:\Users\tholl\Desktop\Coding\bio alg project>python primer2.py
Checkpoint 1/5 Reached: 0.1830136775970459
Checkpoint 2/5 Reached: 0.0029854774475097656
Checkpoint 3/5 Reached: 6.142516374588013
Checkpoint 4/5 Reached: 31.09688949584961 check point 5 can take 1343 seconds to finish
Compared against sequence 0 / 9735 Seconds taken: 2.77089524269104
Compared against sequence 1 / 9735 Seconds taken: 1.228081464767456
Compared against sequence 2 / 9735 Seconds taken: 1.9174354076385498
Compared against sequence 3 / 9735 Seconds taken: 1.009289264678955
Compared against sequence 4 / 9735 Seconds taken: 0.9903068542480469
Compared against sequence 5 / 9735 Seconds taken: 1.0328478813171387
Compared against sequence 6 / 9735 Seconds taken: 1.2636380195617676
Compared against sequence 7 / 9735 Seconds taken: 1.160902976989746
Compared against sequence 8 / 9735 Seconds taken: 1.0072722434997559
Compared against sequence 9 / 9735 Seconds taken: 1.3719615936279297
Compared against sequence 10 / 9735 Seconds taken: 0.8482565879821777
Compared against sequence 11 / 9735 Seconds taken: 0.8147835731506348
Compared against sequence 12 / 9735 Seconds taken: 0.9401898384094238
Compared against sequence 13 / 9735 Seconds taken: 0.8761303424835205
Compared against sequence 14 / 9735 Seconds taken: 0.7868862152099609
Compared against sequence 15 / 9735 Seconds taken: 0.7260477542877197
Compared against sequence 16 / 9735 Seconds taken: 0.7637856006622314
Compared against sequence 17 / 9735 Seconds taken: 0.8679614067077637
```

The ranked subsequence list is cleaned up to remove the subsequences that would be too degenerate due to there not being enough matches between the primer and sequences (0 matches to 13). This cleaned up ranking is then compared with each of the other subsequence lists generated for the other wanted sequence lists and it is cleaned up (removed subsequences that are too degenerate) after each list is processed. When this is completed, we have finished checkpoints 1- 4 (shown in the picture above).

In checkpoint 5 the program takes the ranked subsequence list and compares it against all of the unwanted sequences. It does this by creating a list of all possible subsequences for each sequence in this list. It then compares this list of subsequences with the ranked subsequences to rerank them. After they are reranked the program removes the sequences that attached to the unwanted sequence (had 16-18 matches). After each of the sequences is checked the program prints out its indices and the time it took so that the user can see that the program is proceeding.



```
Compared against sequence 9718 / 9735 Seconds taken: 0.0009968280792236328
Compared against sequence 9719 / 9735 Seconds taken: 0.0009980201721191406
Compared against sequence 9720 / 9735 Seconds taken: 0.001001596450805664
Compared against sequence 9721 / 9735 Seconds taken: 0.0
Compared against sequence 9722 / 9735 Seconds taken: 0.0009989738464355469
Compared against sequence 9723 / 9735 Seconds taken: 0.0
Compared against sequence 9724 / 9735 Seconds taken: 0.0010137557983398438
Compared against sequence 9725 / 9735 Seconds taken: 0.0
Compared against sequence 9726 / 9735 Seconds taken: 0.0
Compared against sequence 9727 / 9735 Seconds taken: 0.0009992122650146484
Compared against sequence 9728 / 9735 Seconds taken: 0.0
Compared against sequence 9729 / 9735 Seconds taken: 0.0010035037994384766
Compared against sequence 9730 / 9735 Seconds taken: 0.0009951591491699219
Compared against sequence 9731 / 9735 Seconds taken: 0.0
Compared against sequence 9732 / 9735 Seconds taken: 0.0009958744049072266
Compared against sequence 9733 / 9735 Seconds taken: 0.0
Compared against sequence 9734 / 9735 Seconds taken: 0.0009984970092773438
Checkpoint 5/5 Reached: 2081.091356754303
There are no subsequences that will attach to the wanted set but not the unwanted set. The set that will attach to the wanted set and will
 also attach to the unwanted set is :
GCGACCCCGCAAGGGGGA
AATGGGAAGCGACCCCGC
AATCGGTAGTAACGGGGT
TAGTAACGGGGTATCAGC
TCCCAGAAACCCGGTCAT
CCCAGAAACCCGGTCATG
CCAGAAACCCGGTCATGG
AGAAACCCGGTCATGGTG
ACAATGGGAAGCGACCCC
CAATGGGAAGCGACCCCG
GGGGCTGAAACTCGCCCC
TCCCCGTGAATACGTTCT
GTGCGGATTGGGGGCTGA
TGCGGATTGGGGGCTGAA
GCGGATTGGGGGCTGAAA
GTCCCCGTGAATACGTTC
GGTGCGGATTGGGGGCTG
TAATACCCCATAAAGCCG
GAAACCCGGTCATGGTGC
AAACCCGGTCATGGTGCG
AACCCGGTCATGGTGCGG
ACCCGGTCATGGTGCGGA
CCCGGTCATGGTGCGGAT
CCGGTCATGGTGCGGATT
CGGTCATGGTGCGGATTG
CGGGCAGGCGGAATTCCC
CTTGACGGTACCCCCAGA
CCGGCCGTAAACCATGGG
GGCCGTAAACCATGGGCG
```

After this is finished the program prints out the valid subsequences that attach to the wanted set of sequences and not the unwanted set. If no subsequences are left after comparing with the unwanted set the program prints out the subsequences that could be used to attach to the wanted set that would also attach to the wanted set.

When none are left it means that finding a degenerate primer that attaches to the subset and not the whole set is not possible for this set of sequences. This happens with this data set and that makes sense as are they are all sequences from similar bacteria.

Future work

Future work would include:

- Creating a degenerate primer from the list of subsequences produced and attempting to attach it to the unwanted set instead of using a ranking comparison
- Replacing the arrays with a better data structure. Will discuss this in a meeting with you after the break. I have been looking into whether there might be a more optimal data structure such as a modified tree to use for comparison between the sequences. One option may be a modified trie but based on my research I think that a modified set of automata's may provide the best results.

  If there is an automata model made for each sequence, then the program can hand off each comparison of two sequences to a thread that returns the area with the best match. This would be more optimal because the construction of the automata would be slow but once they are made, they would help to increase the speed of comparisons. These modified automata would have to add a degenerate link to the primer whenever there is a mismatch. I am not entirely sure if this is possible and so I hope to meet with you to discuss it after the thanksgiving break.
- Can change the subsequence creation size to be larger then the minimum of 18. Degenerate primers might be possible

Bibliography

[1] T. M. Rose, J. G. Henikoff, and S. Henikoff, “CODEHOP (COnsensus-DEgenerate Hybrid Oligonucleotide Primer) PCR primer design,” *Nucleic Acids Res.*, 2003.

[2] R. Boyce, P. Chilana, and T. M. Rose, “iCODEHOP: A new interactive program for designing COnsensus-DEgenerate Hybrid Oligonucleotide Primers from multiply aligned protein sequences,” *Nucleic Acids Res.,* 2009.

[3] C. Linhart and R. Shamir, “The degenerate primer design problem,” *Bioinformatics,* 2002.

[4] X. Wei, D. N. Kuhn, and G. Narasimhan, “Degenerate primer design via clustering,” in *Proceedings of the 2003 IEEE Bioinformatics Conference, CSB 2003,* 2003.

[5] J. Ye, G. Coulouris, I. Zaretskaya, I. Cutcutache, S. Rozen, and T. L. Madden, “Primer-BLAST: a tool to design target-specific primers for polymerase chain reaction.,” *BMC Bioinformatics,* 2012.

[6] A. Untergasser *et al.*, “Primer3-new capabilities and interfaces,” *Nucleic Acids Res.,* 2012.