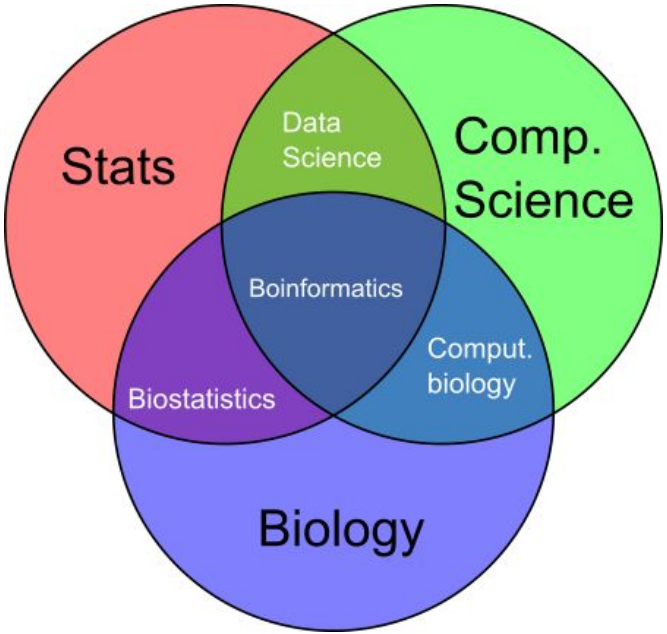
MMORE-MMATCHES: A Heuristic Forward-Backward

David Rich

Bioinformatics

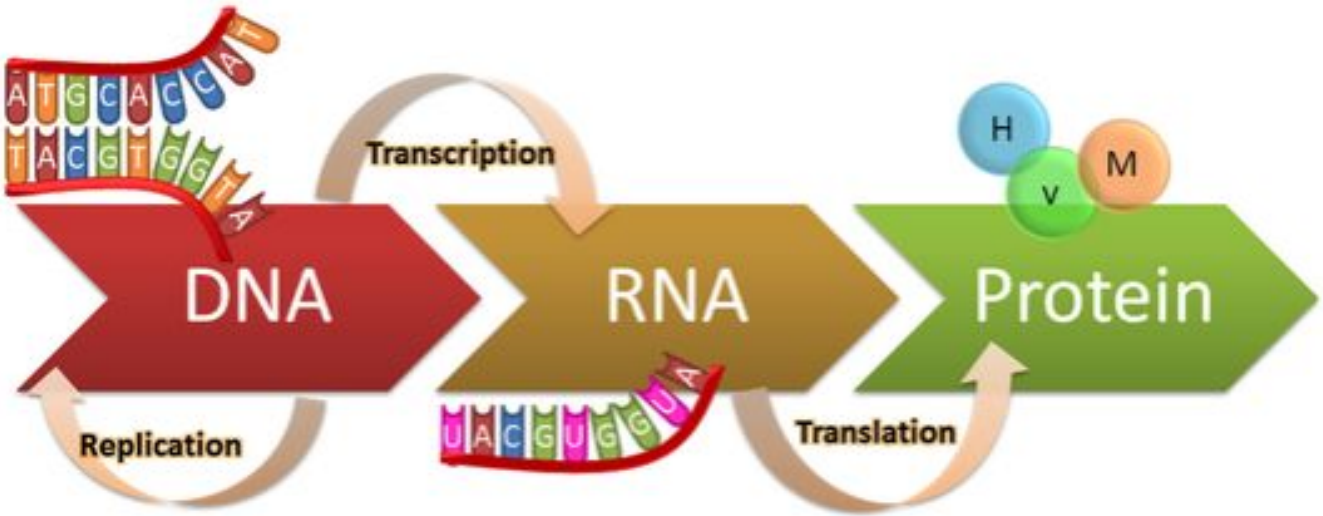
What is Bioinformatics?

- Bioinformatics is the interdisciplinary science of computer science, biology, and statistics.
-

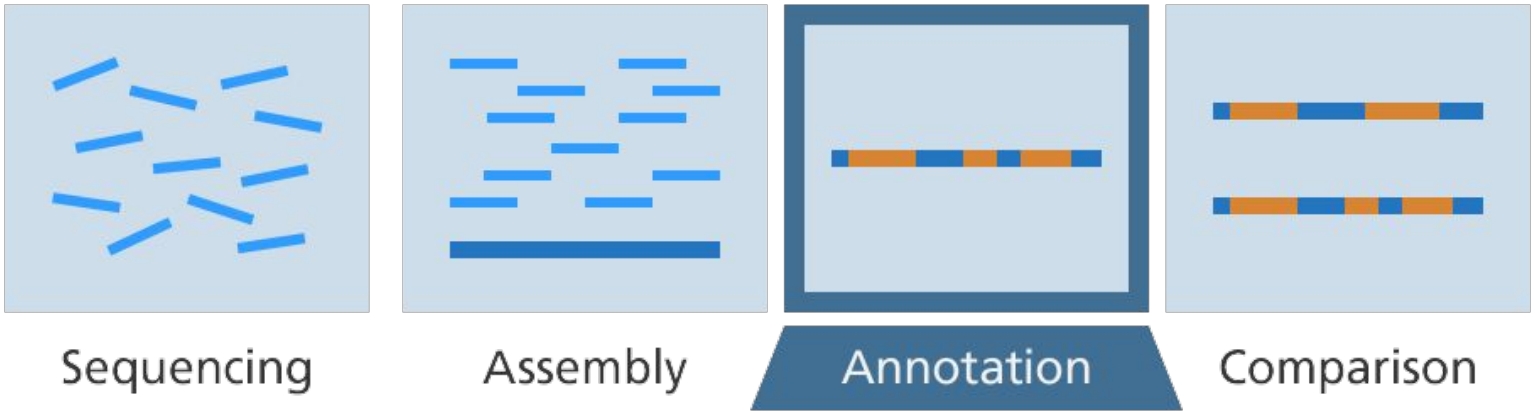


Sequencing vs. Analysis

Central Dogma of Molecular Biology

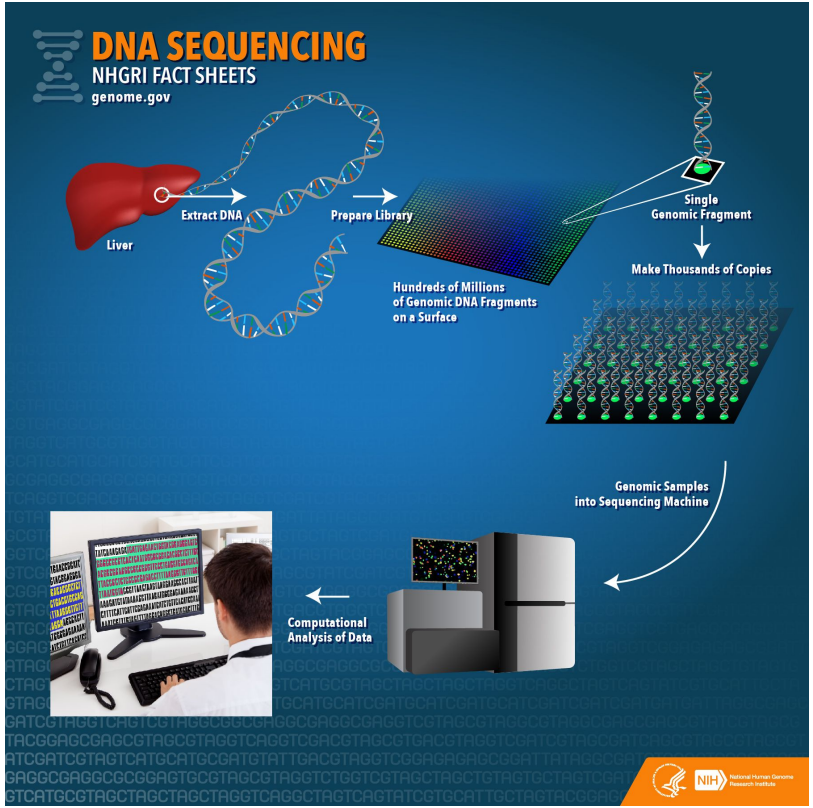


Sequencing vs. Analysis



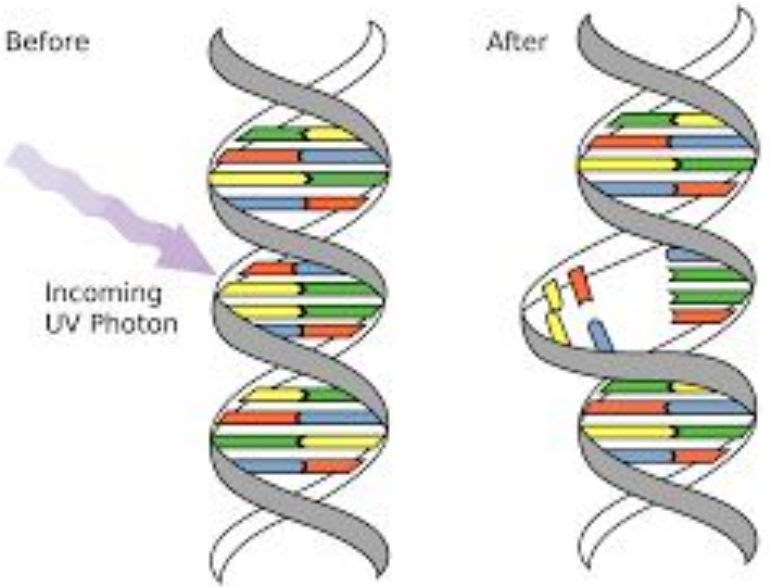
Sequencing vs. Analysis

- **DNA Sequencing** is the process of determining the ordering of genetic sequence.
- **DNA Annotation** is the process of determining the location and function of genes within a greater genetic sequence.
- **Molecular Phylogeny** is the process of using DNA to find the hereditary relationships between organisms.
- Once a genome is sequenced, other processes must take place in order to make sense of it.



DNA Mutations

- Mutation are the substitutions and indels into DNA. These occur naturally over time.
- Because of this, alignment similarity can tell us how distantly related two things are.
- This can also show us important parts of the genome, as selective pressure causes regions integral to survival to be highly conserved over time and across species.



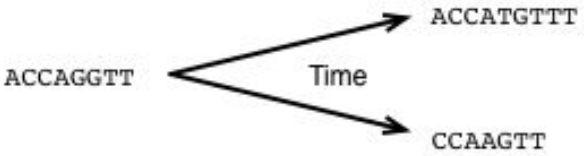
Sequencing vs. Analysis

- There has been a four order magnitude decrease in the cost of sequencing since 2007.
- The throughput of DNA sequencing has exceeded the pace of computational speedup. More and more, this has caused analysis to become the bottleneck in the pipeline of DNA workflow.
- Due to this, it has become increasingly important to have fast and accurate methods for analyzing genetic sequences.

Sequence Alignment

Sequence Alignment

- **Sequence Alignment** is a central task in the process of annotation and many other research goals.
- Sequence Alignment shows the similarity of two sequences by finding the likeliest probable alignment between them through a series of matches, substitutions and indels (insertions and deletions).



A C C A T G T T T
– C C A A G – T T

Smith-Waterman

- One of the first algorithms used for solving the problem of sequence alignment.
- Finds the best sequence-to-sequence alignment by finding maximal path through dynamic programming matrix.
- Uses constant terms for its match, insert, and delete scoring scheme.
- Each path through an alignment describes a series of insertions and deletions that explains how a given query could've produced a given target sequence through a series of mutations.
- Each score describes the likelihood of this event occurring.

	\	A	T	C	T	C	G	T	A	T	G	A	T
\	0	-2	-4	-6	-8	-10	-12	-14	-16	-18	-20	-22	-24
G	-2	-1	-2	-3	-4	-5	-6	-7	-8	-9	-10	-11	-12
T	-4	-2	1	0	-1	-2	-3	-4	-5	-6	-7	-8	-9
C	-6	-3	0	3	2	1	0	-1	-2	-3	-4	-5	-6
T	-8	-4	-1	2	5	4	3	2	1	0	-1	-2	-3
A	-10	-5	-2	1	4	4	3	2	4	3	2	1	0
T	-12	-6	-3	0	3	3	3	5	4	6	5	4	3
C	-14	-7	-4	-1	2	5	4	4	4	5	5	4	3
A	-16	-8	-5	-2	1	4	4	3	6	5	4	7	6
C	-18	-9	-6	-3	0	3	3	3	5	5	4	6	6

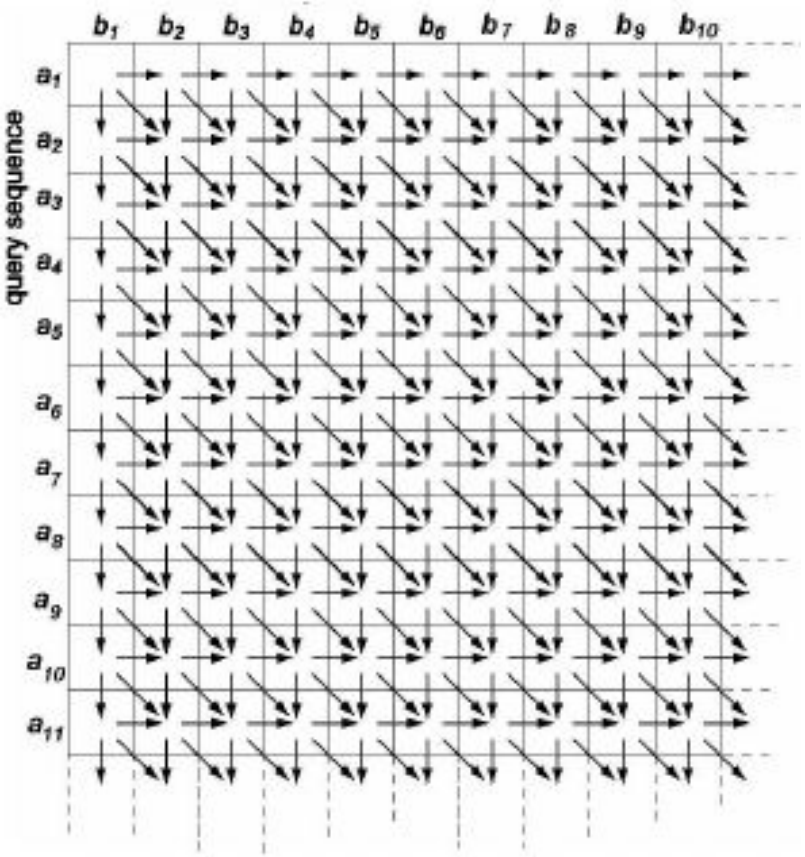
The best global alignment would be:

where "|" = match ; "-" = gap

```
ATCTCGTATGAT
|| |||  |
G--TC-TATCAC
```

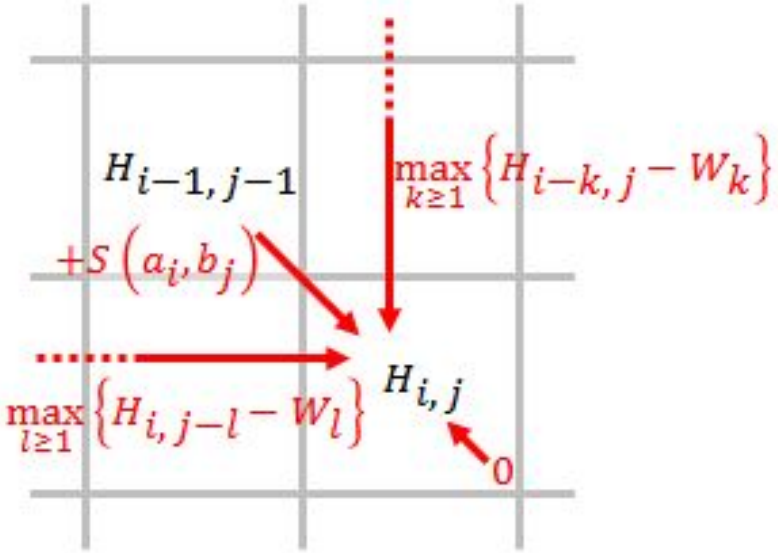
Smith-Waterman: Runtime

- Every cell is conditionally dependent upon all cells to it left and right.
- By calculating these cells from top-left to bottom-right, we can isolate this relationship such that each cell is only dependent on its immediately adjacent cells.



Smith-Waterman: Runtime

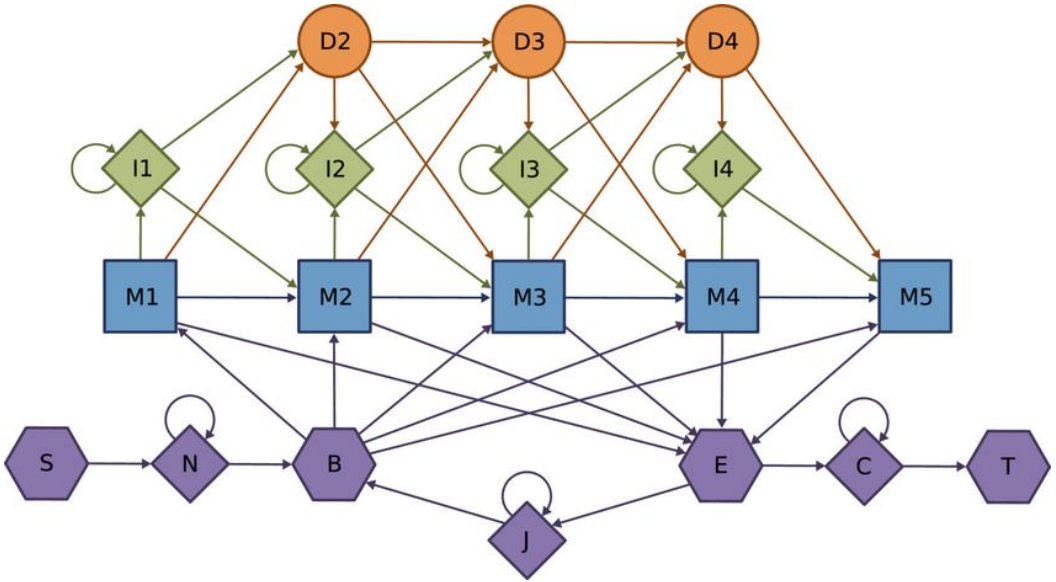
- If both sequences are of length ~N, then there will ~N^2 cells in the dynamic programming matrix (H).
- If we move through the matrix row-by-row, at every cell H(i,j), we will have already computed the values of its dependent cells.
- Therefore, even though there are far more than N^2 possible alignments, we only need to compute each cell once.



Hidden Markov Models, Viterbi & Forward-Backward

Hidden Markov Model

- Each edge in the graph has a given probability for determining the likelihood of reaching that point in the graph (given the sequences are related).
- This more allows us a much more robust model of the sequences (or sequence families). We can have unique match, insert, and delete probabilities based on position, as well as account for background frequencies of matches.



Hidden Markov Model vs. Sequence

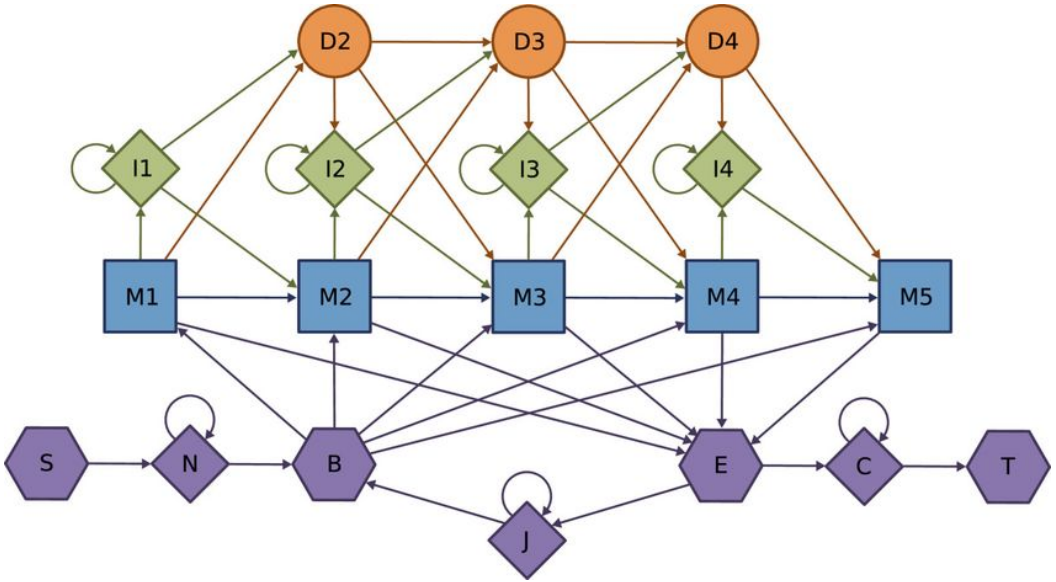
- Sequences
  - Sequences do not have position specific scoring. All cells are discretely determined to be one amino acid.
  - If we are uncertain, this cannot be reflected in a sequence.

- Hidden Markov Model
  - HMM's allow for uncertainty.
  - Each position has a probability to be a particular amino acid.
  - This allows us to integrate over this uncertainty.

```
HMMER3/f [3.2.1 | June 2018]
NAME  test1_2
LENG  15
ALPH  amino
RF    no
MM    no
CONS  yes
CS    no
MAP   yes
DATE  Thu Jan  2 16:31:59 2020
NSEQ  1
EFFN  1.000000
CKSUM 3765312231
STATS LOCAL MSV       -6.5511  0.75512
STATS LOCAL VITERBI   -6.7900  0.75512
STATS LOCAL FORWARD   -3.0531  0.75512
HMM          A        C        D        E        F        G        H        I        K        L        M        N        P        Q        R        S        T        V        W        Y
           m->m     m->i     m->d     i->m     i->i     d->m     d->d
  COMPO   3.26796  5.00778  2.80104  3.34084  2.80139  3.68621  3.00898  3.61474  2.61564  2.58084  3.24822  3.68420  2.86683  2.20762  2.30549  2.77273  3.58636  3.47705  5.01705  2.40837
          2.68618  4.42225  2.77519  2.73123  3.46354  2.40513  3.72494  3.29354  2.67741  2.69355  4.24690  2.90347  2.73739  3.18146  2.89801  2.37887  2.77519  2.98518  4.58477  3.61503
          0.02276  4.18987  4.91222  0.61958  0.77255  0.00000        *
      1   3.40184  5.19446  3.92838  3.38722  4.63204  3.81259  4.10738  4.20674  2.43586  3.66497  4.66611  3.71718  4.30889  3.32078  0.58595  3.47711  3.65193  3.91932  5.59128  4.50174
          2.68618  4.42225  2.77519  2.73123  3.46354  2.40513  3.72494  3.29354  2.67741  2.69355  4.24690  2.90347  2.73739  3.18146  2.89801  2.37887  2.77519  2.98518  4.58477  3.61503
          0.02276  4.18987  4.91222  0.61958  0.77255  0.48576  0.95510
      2   3.73833  5.09708  4.32357  4.12283  2.38109  4.21303  3.81570  3.70781  3.97247  3.07660  4.35100  4.13723  4.68411  4.19909  4.08593  3.83855  4.02918  3.59463  4.03332  0.56656
          2.68618  4.42225  2.77519  2.73123  3.46354  2.40513  3.72494  3.29354  2.67741  2.69355  4.24690  2.90347  2.73739  3.18146  2.89801  2.37887  2.77519  2.98518  4.58477  3.61503
          0.02276  4.18987  4.91222  0.61958  0.77255  0.48576  0.95510
```

Viterbi Algorithm

- The Viterbi algorithm is a generalization of the Smith-Waterman algorithm.
- The difference is that Smith-Waterman relates sequence-to-sequence, where Viterbi is profile-to-sequence, which includes position-specific scoring.
- At each state, we can select the maximal probability score of its previous state plus its incoming edge (transition probability).

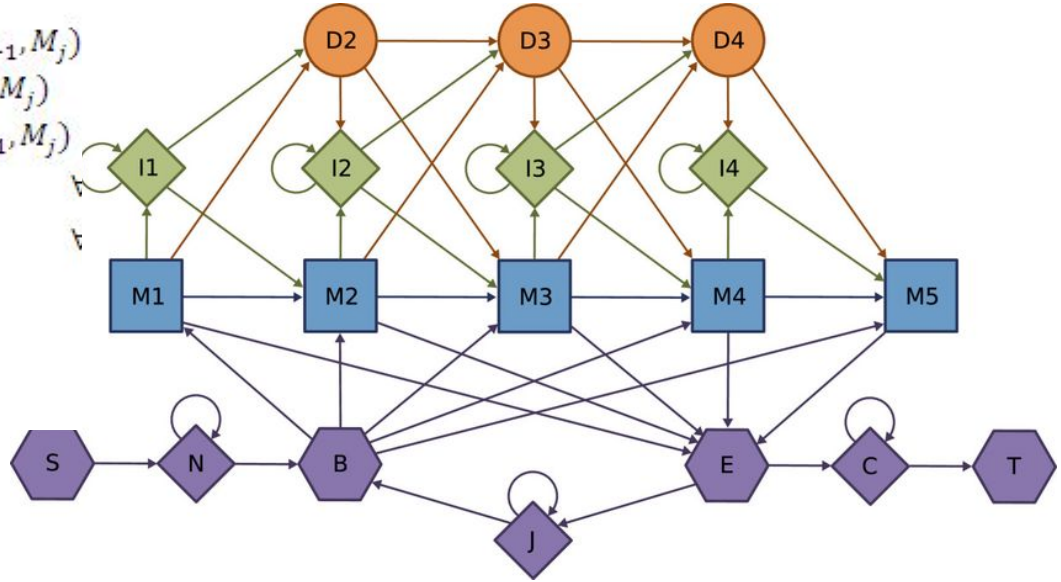


Viterbi Algorithm

$$M(i,j) = em(M_j, s_i) + max \begin{cases} M(i-1,j-1) + tr(M_{j-1}, M_j) \\ I(i-1,j-1) + tr(I_{j-1}, M_j) \\ D(i-1,j-1) + tr(D_{j-1}, M_j) \\ B(i-1) + tr(B, M_j) \end{cases}$$

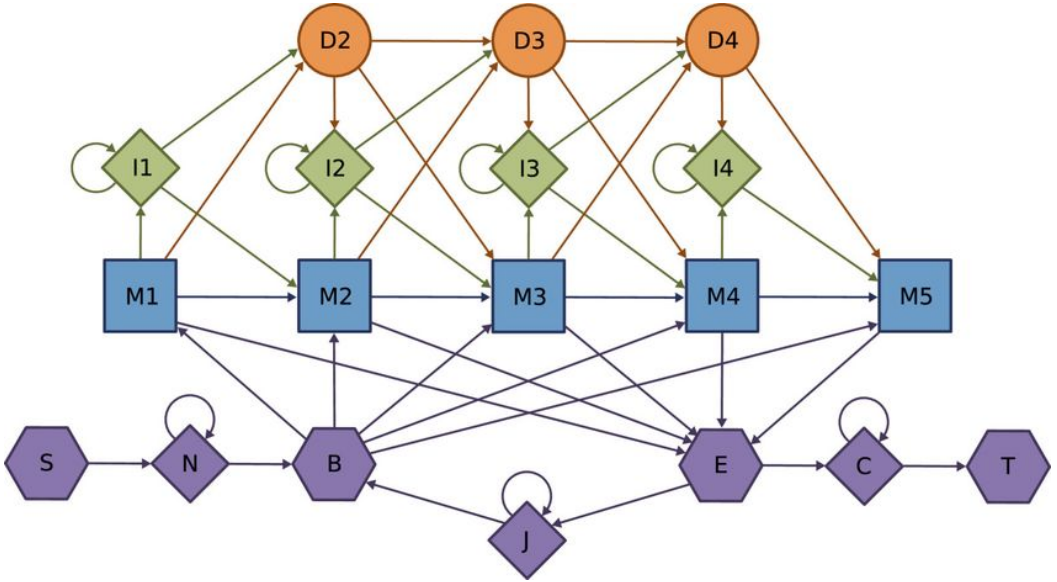
$$I(i,j) = em(I_j, s_i) + max \begin{cases} M(i-1,j) + tr(M_j, I_j) \\ I(i-1,j) + tr(I_j, I_j) \end{cases}$$

$$D(i,j) = max \begin{cases} M(i,j-1) + tr(M_{j-1}, D_j) \\ D(i,j-1) + tr(D_{j-1}, D_j) \end{cases}$$



Viterbi: Runtime

- For its implementation, Viterbi uses 3 dynamic programming matrices (for Match, Insert, and Delete states).
- Since it has to load the position-specific probabilities, it generally takes longer than a simple Smith-Waterman.



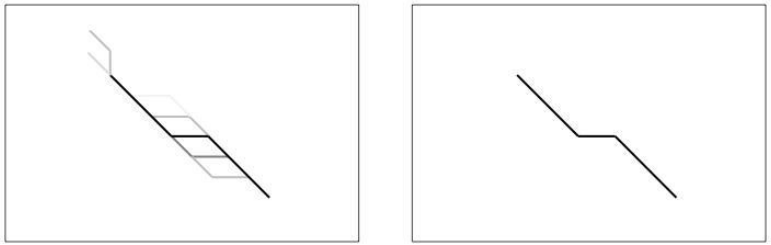
Viterbi vs. Forward: Paths

- A weakness of Viterbi is that it only accounts for the maximal path through the HMM. This can leave a large amount of the support for a potential alignment unaccounted for. This lost data can obscure the level of similarity between two possible sequences.
- The Forward-Backward Algorithm solves this issue by summing over all possible paths through the model.

sum of alignments

approximation

$$F = \log \frac{\Sigma_\pi P(t,\pi \mid H)}{P(t \mid R)} \qquad V = \log \frac{P(t,\pi^o \mid H)}{P(t \mid R)}$$



Why integrate over alignments?

```
agaccaaact
agaccaact

agaccaaact    agaccaaact    agaccaaact
agacc-aact    agacca-act    agaccaa-ct
```

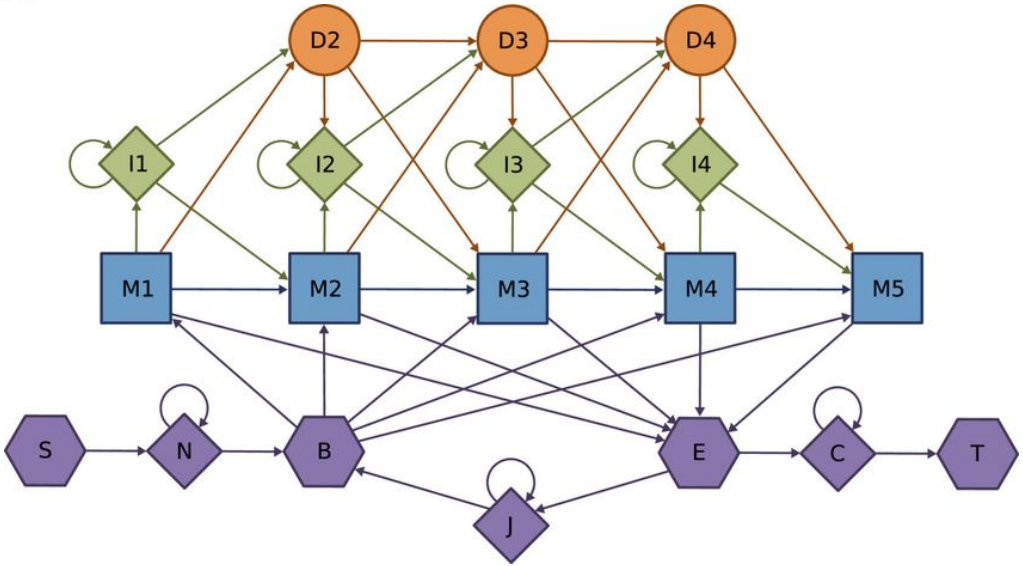
Forward Algorithm

$$M(i,j) = em(M_j, s_i) + sum \begin{cases} M(i-1,j-1) + tr(M_{j-1}, M_j) \\ I(i-1,j-1) + tr(I_{j-1}, M_j) \\ D(i-1,j-1) + tr(D_{j-1}, M_j) \\ B(i-1) + tr(B, M_j) \end{cases}$$

$$I(i,j) = em(I_j, s_i) + sum \begin{cases} M(i-1,j) + tr(M_j, I_j) \\ I(i-1,j) + tr(I_j, I_j) \end{cases}$$

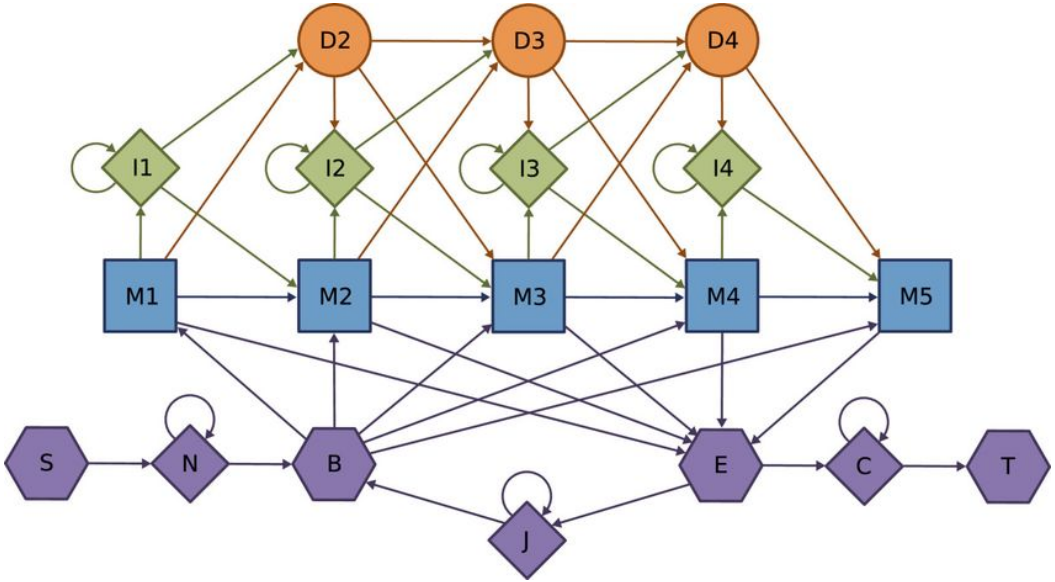
$$D(i,j) = sum \begin{cases} M(i,j-1) + tr(M_{j-1}, D_j) \\ D(i,j-1) + tr(D_{j-1}, D_j) \end{cases}$$

- Probabilities are stored in log-space, so multiplication of probabilities becomes addition.



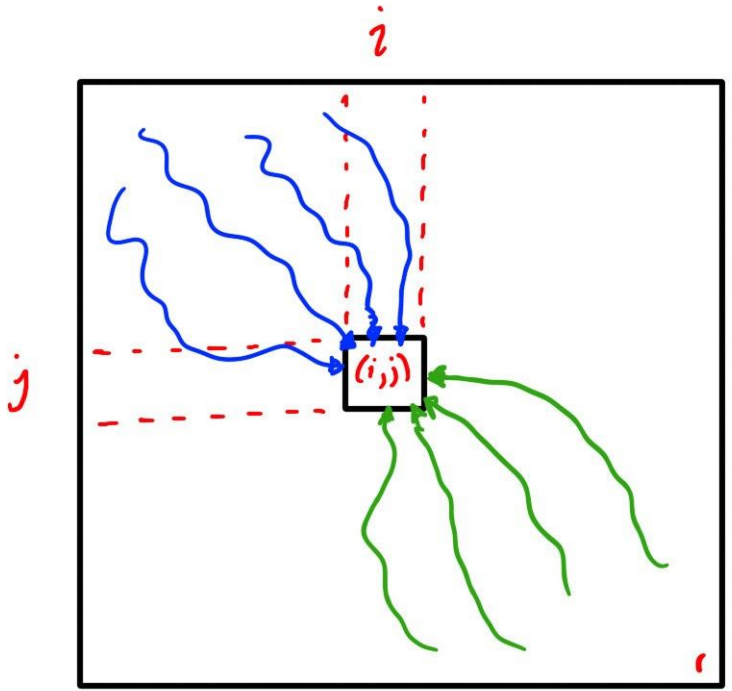
Runtime: Viterbi vs Forward

- There is a major increase in the runtime of Forward-Backward over Viterbi.
- Probabilities must be stored in log-space, as multiplication of many small probabilities quickly causes underflow even in high precision data-types.
- Because there is no addition under log-space, this entails an expensive process of jumping between logarithmic and linear scales and/or factor scaling of probabilities.



Forward-Backward Alignment Recovery

- We can recover the alignment by recovering the Posterior Probability.
- This means we can essentially multiply the Forward and Backward dp matrices together in order to obtain the Posterior.
- Then the highest likelihood path in the Posterior is the alignment.



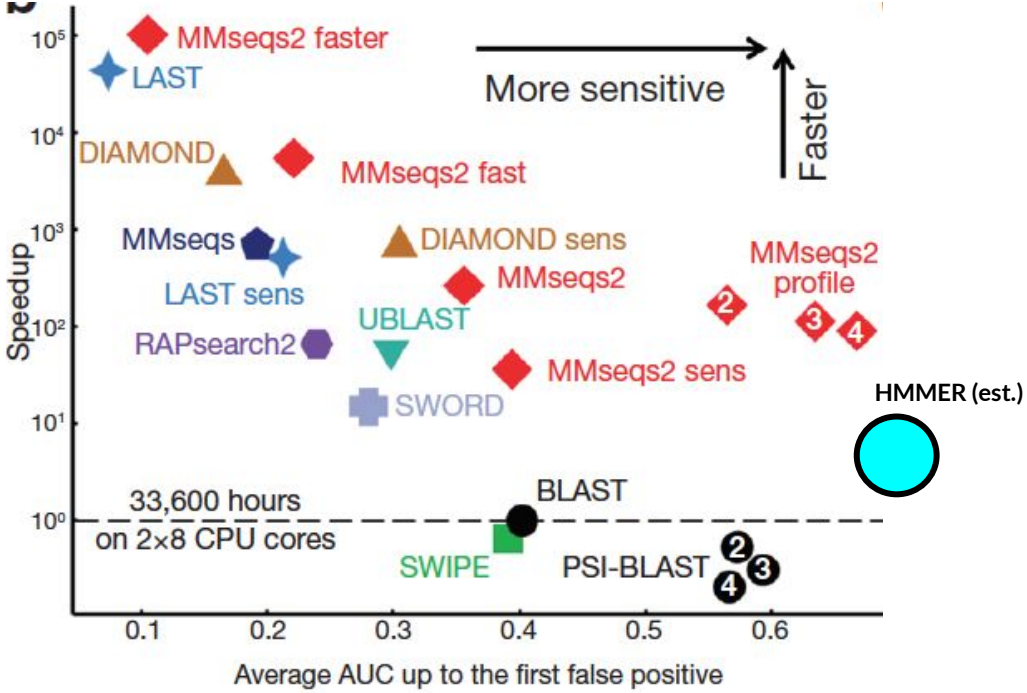
HMMER
& MMSEQS

HMMER vs MMSEQS

- HMMER and MMSEQS are two of the more popular sequence alignment software suites.
- HMMER generally outperforms MMSEQS in accuracy.
- MMSEQS is generally faster than HMMER.

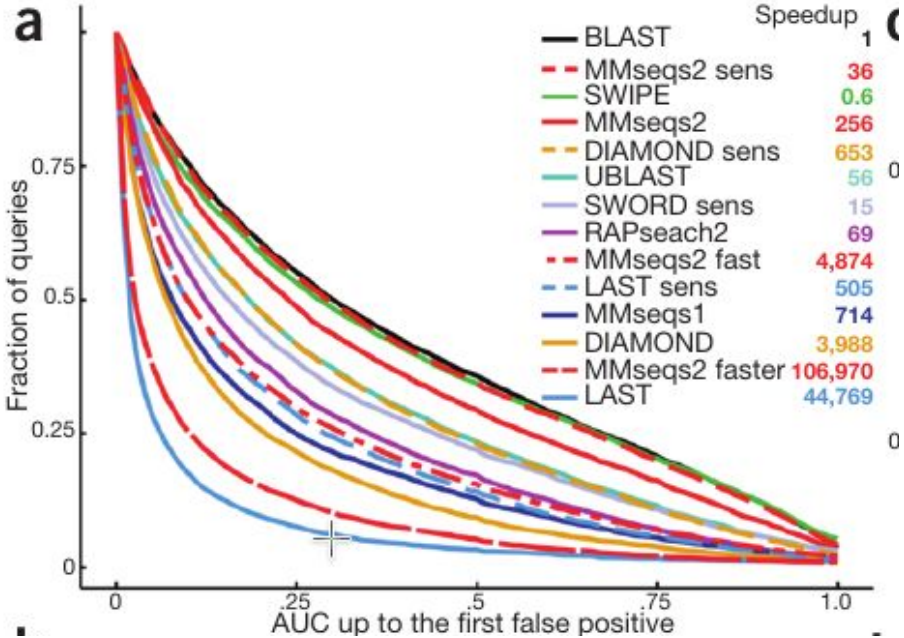
HMMER vs MMSEQS

- HMMER and MMSEQS are two of the more popular sequence alignment software suites.
- HMMER generally outperforms MMSEQS in accuracy.
- MMSEQS is generally faster than HMMER.
- AUC1 = ‘Area Under the Curve 1’ measures the percent of True Positives recovered before the first False Positive.

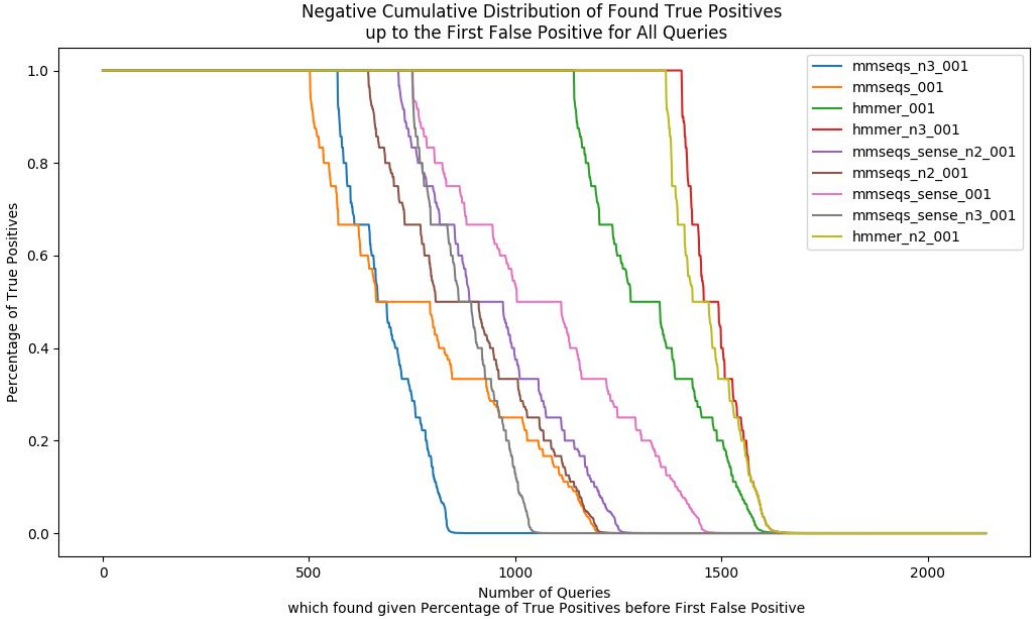


HMMER vs MMSEQS

- HMMER and MMSEQS are two of the more popular sequence alignment software suites.
- HMMER generally outperforms MMSEQS in accuracy.
- MMSEQS is generally faster than HMMER.
- AUC1 = ‘Area Under the Curve 1’ measures the percent of True Positives recovered before the first False Positive.

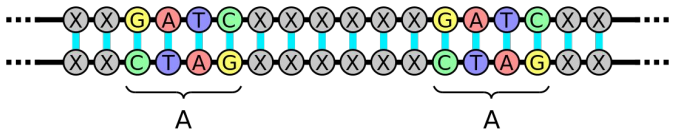


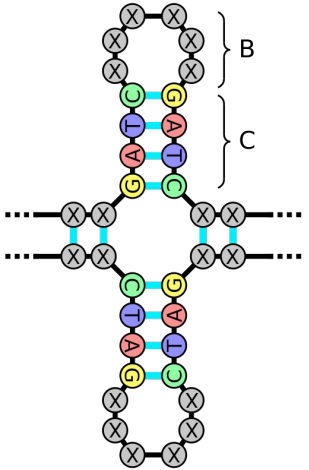
HMMER vs MMSEQS



Issues with MMseqs Benchmark: Reversed vs Shuffled

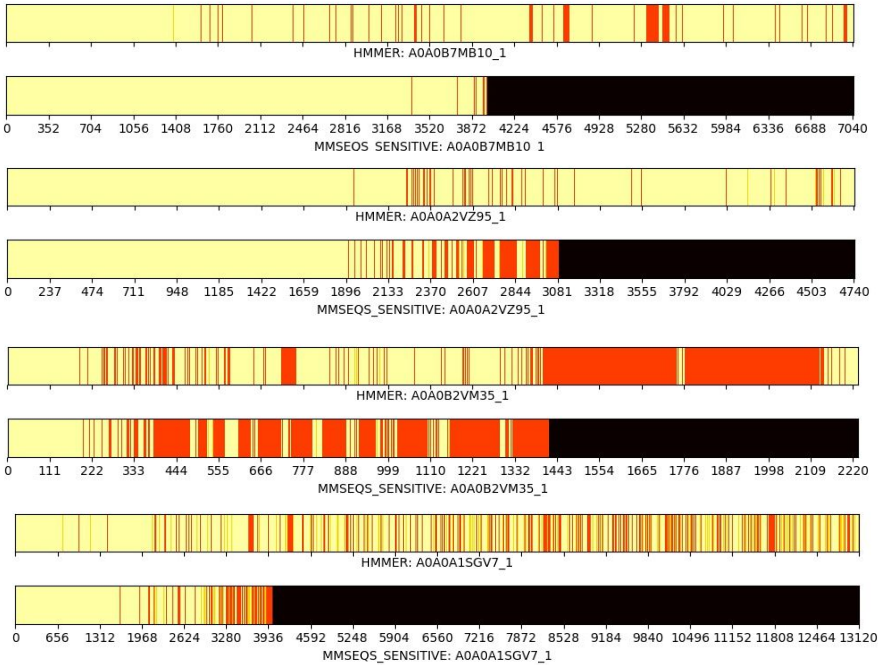
- The benchmark in MMseqs2's paper, it uses a dataset composed from sequences in the UniProt database. For the decoy sequences, it used reverse sequences.
- The issue is the presence of palindromic genetic sequences occur at a higher than random rate in nature. This means that a number of FPs found in the MMseqs dataset may actually be picking up on a true signal, resulting in a misleading FP rate when using higher sensitivity tools. (This topic is currently being investigated by Travis).
- For our benchmark, we built it using Profmark to generate our dataset, which generates decoy sequences using shuffled true sequences rather than reversed.





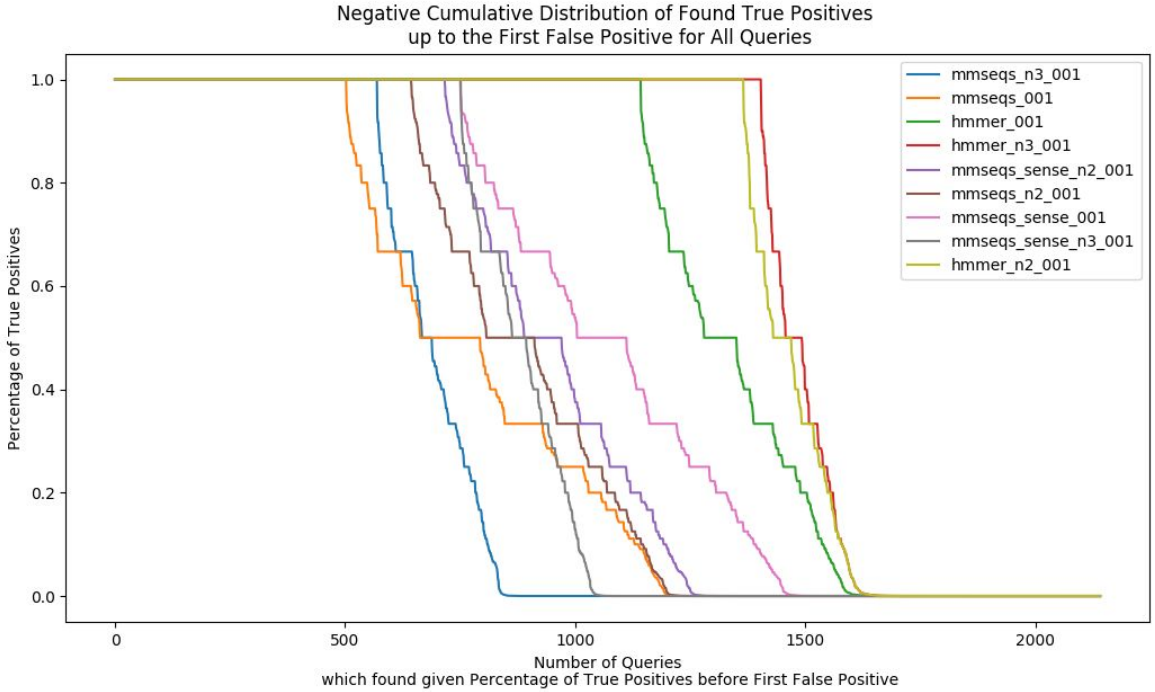
Issues with MMseqs Benchmark: AUC1

- The MMseqs2 paper uses AUC1 as its accuracy measure.
- This strictly appraises the percentage of TPs recovered until the first FP.
- This can be misleading towards the whole picture, as a single early FP can negate a great deal of positive matches that make MMseqs seem superior despite recovering vastly fewer TPs.



HMMER vs MMSEQS: Profmark Results

- Profmark Dataset
- This measures the cumulative total number of queries (N=2141) which recovered a given percent of TP before its first FP.
- Example:
  - HMMER (green), ~1300 queries discovered 50% or more TPs before the first FP.
  - MMSEQS sensitive (pink), ~1100 queries discovered 50% or more TPs before the first FP.
- All MMseqs search modes performed worse than HMMER.

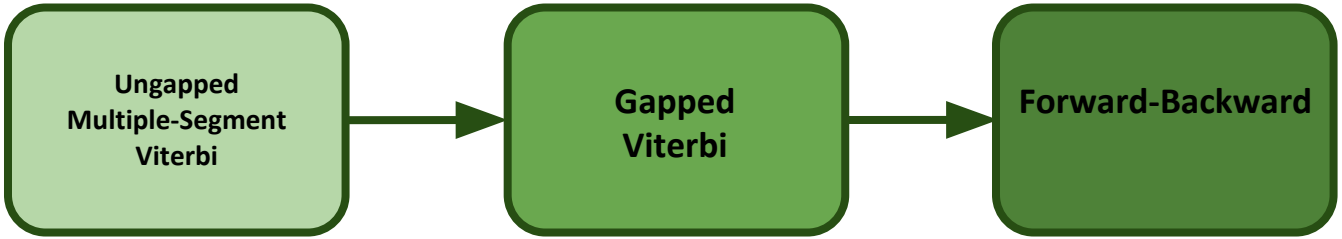


HMMER vs MMSEQS

- Each use a pipeline of increasingly stringent and expensive filtering algorithms to find strong sequence alignments.
- Their pipelines are very similar (both use gapped and ungapped Viterbi), apart from a notable difference: HMMER's use of the Forward-Backward algorithm.
- If we are able to speed up the Forward-Backward algorithm, we hope to capture the speed of MMSEQS and the accuracy of HMMER.

Pipelines

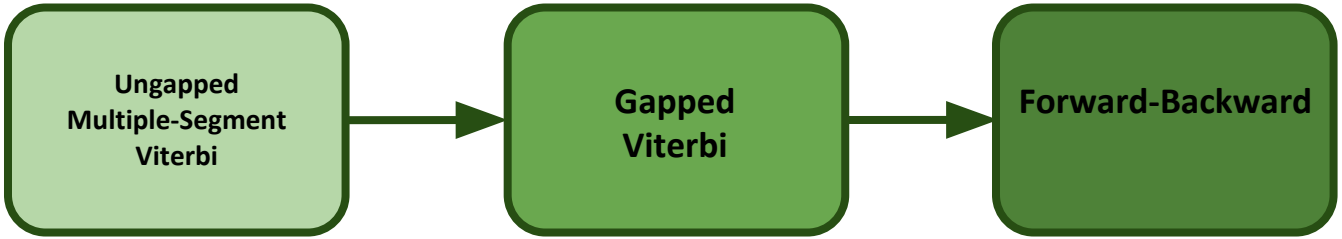
HMMER Pipeline



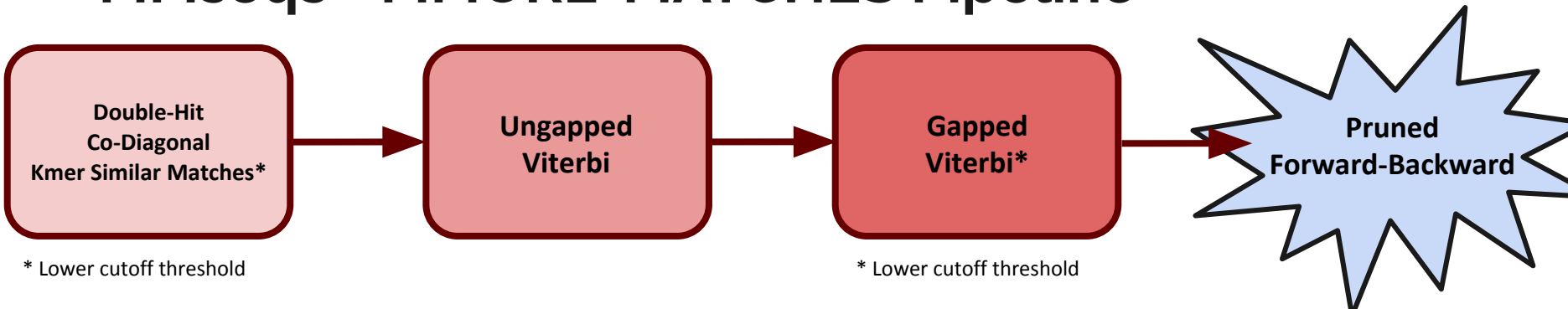
MMseqs Pipeline

Pipelines

HMMER Pipeline

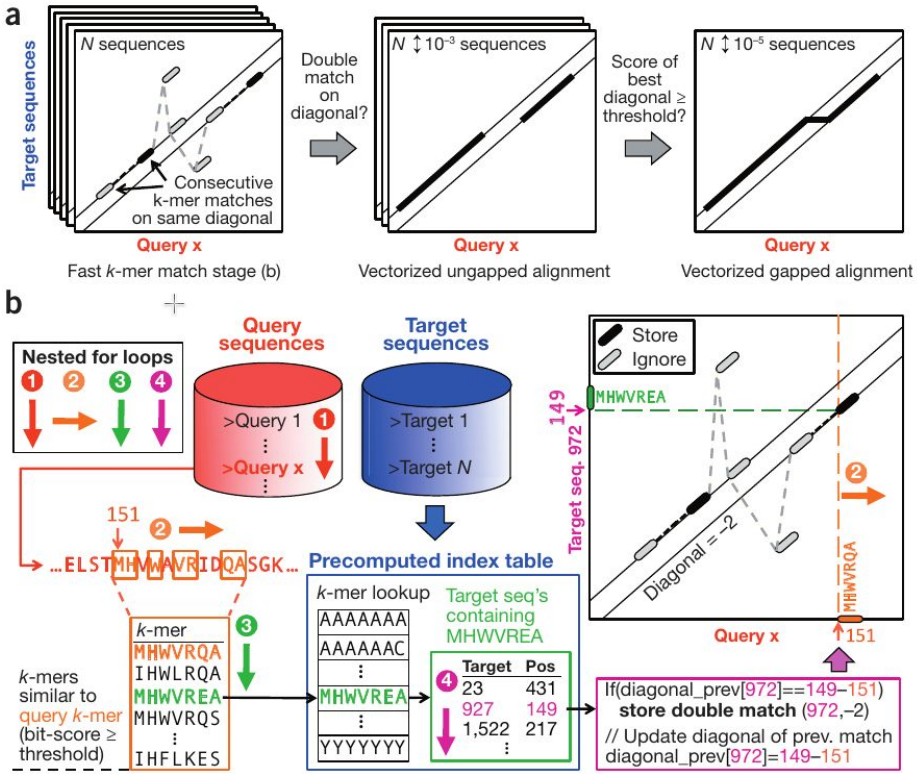


MMseqs + MMORE-MATCHES Pipeline



MMseqs: Double K-mer Prefilter

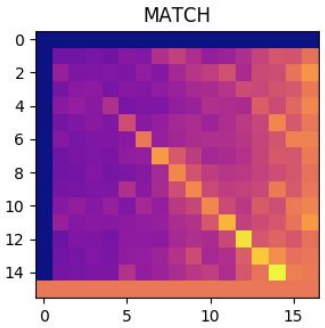
-

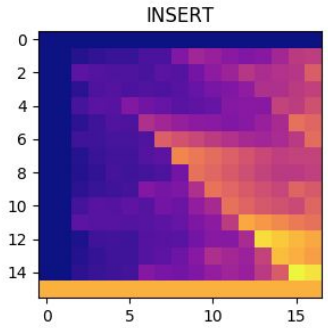


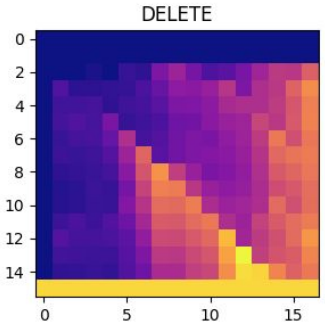
MMORE-MMATCHES: Pruned Forward-Backward

Step 1: Cloud Search

- The goal is reduce the search space. Of the N^2 cells to be computed, many of them do not contribute meaningfully to the final score.
- There are two passes, one in the Forward and one in the Backward direction.

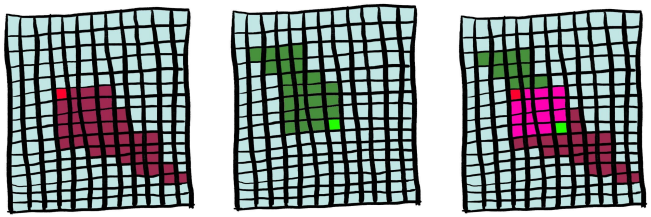




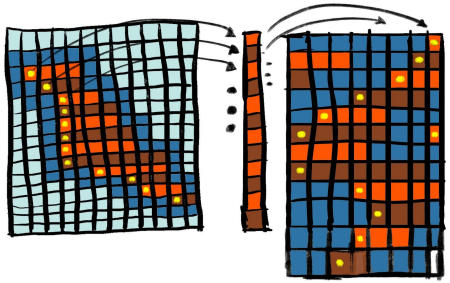


MMORE-MMATCHES: Overview

1. Cloud Search

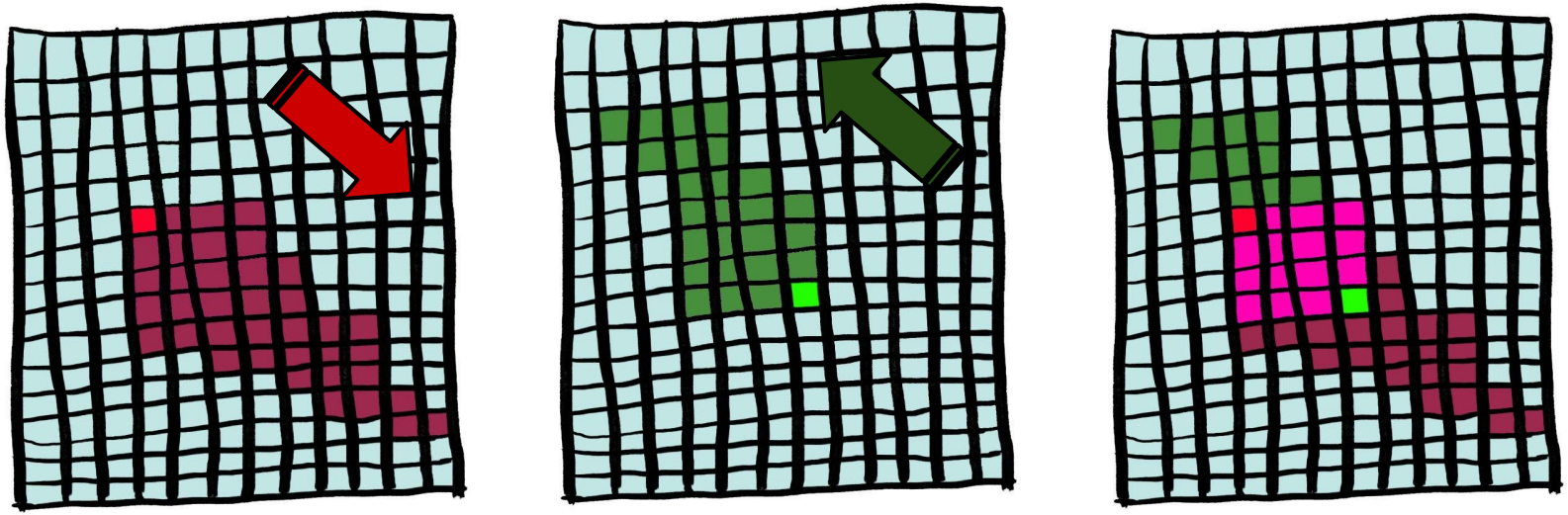


2. Bounded Forward-Backward



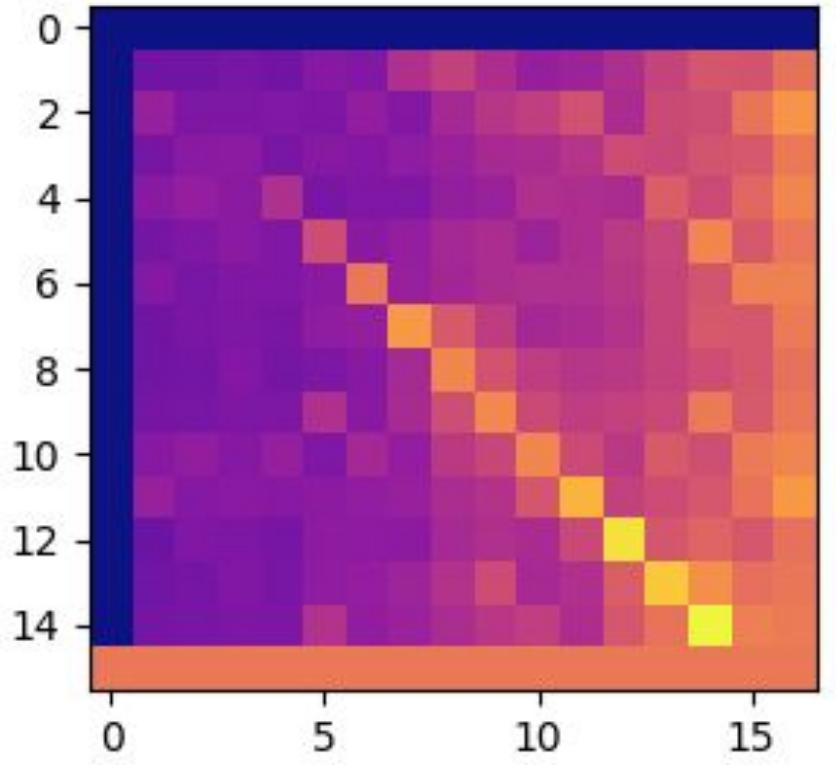
3. Alignment Recovery

Step 1: Cloud Search



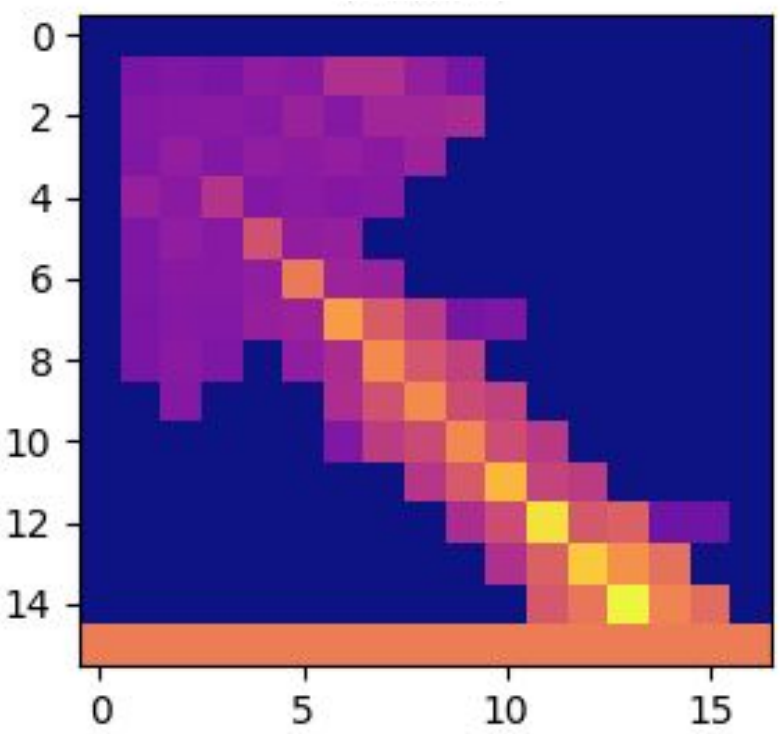
Step 1: Cloud Search

- The goal is reduce the search space. Of the N^2 cells to be computed, many of them do not contribute meaningfully to the final score.
- The Cloud Search step is used to prune the search space to so that only high scoring probability space is computed.



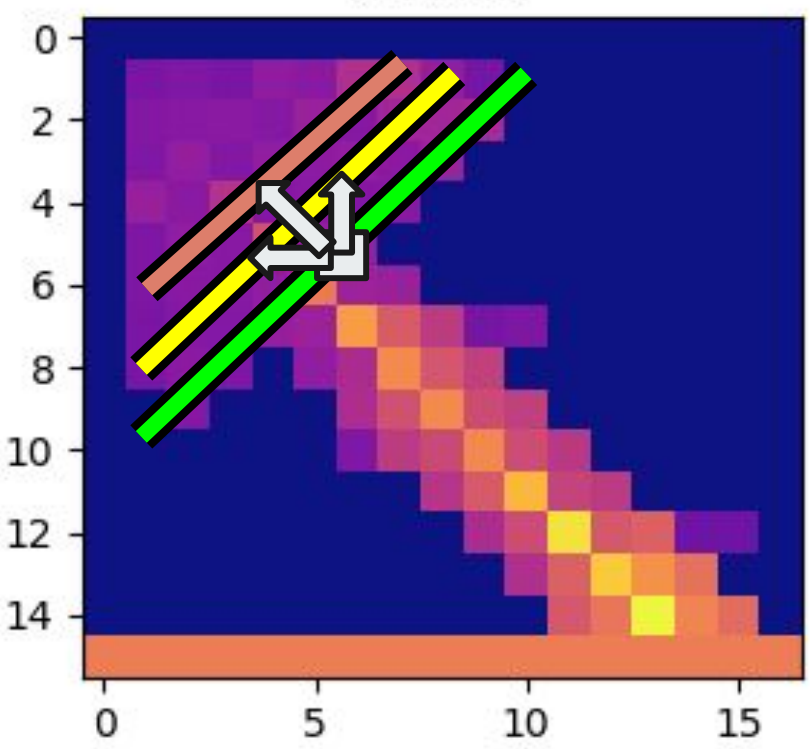
Step 1: Cloud Search

- We begin by using the start and end points of the maximal Viterbi alignment.
- Rather than computing the matrix row-by-row, we sweep anti-diagonally. This ensures that all states equi-distance from start state are in memory simultaneously so they can be compared at the same time.



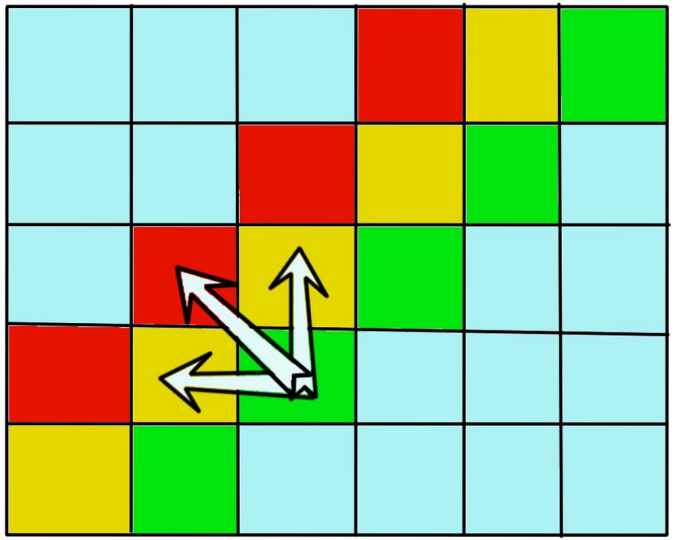
Step 1: Cloud Search

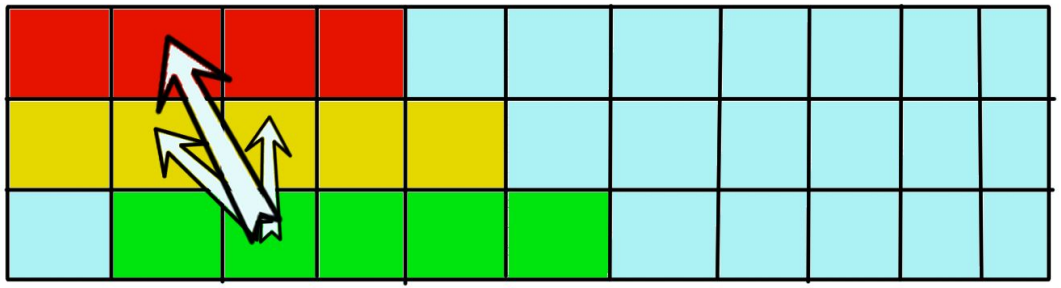
- At each antidiag, all cells adjacent to previous active cells are computed.
- Using the maximum score seen thus far (M) in the entire search, and the maximum score for the current diagonal (m), the antidiag is pruned according to the parameters.
- There are two tuning parameters:
  - Alpha: For each cell in the antidiag, if the cell's score is less than (m - alpha), it is pruned.
  - Beta: For each antidiag, if no cells in the current antidiagonal are above (M - beta) then the search is completed and no further antidiags are searched.



Step 1: Cloud Search

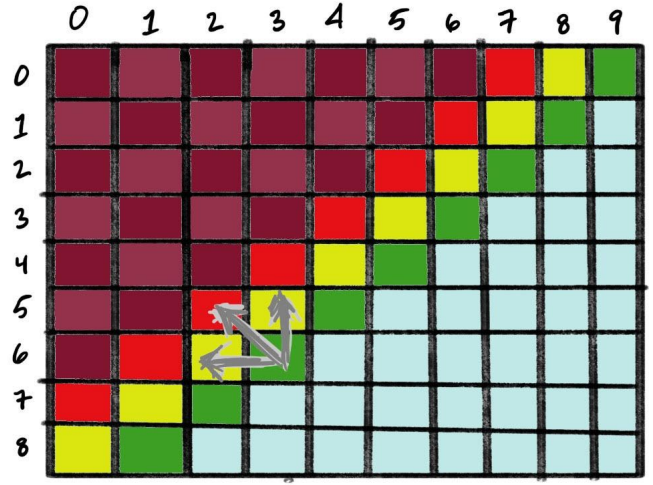
- Traditional Forward-Backward can be done using a [2 * Q] matrix, by storing the data row-wise.
- For Cloud Search, data must be held antidiagonal-wise in order to process them sequentially. This requires [3 * (Q + T)] memory. We require three rows because the diagonal dependency is two anti-diagonals back.
- Antidiagonal indices can be tracked where the sum of the coordinates is equal to the index of its antidiagonal. This creates the example recursion.

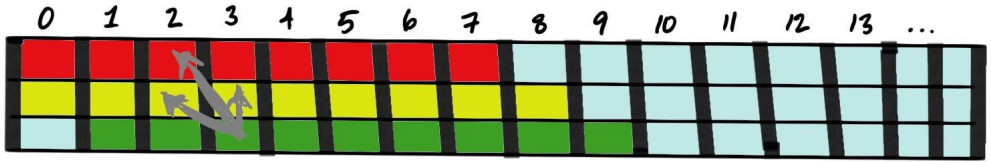




Step 1: Cloud Search

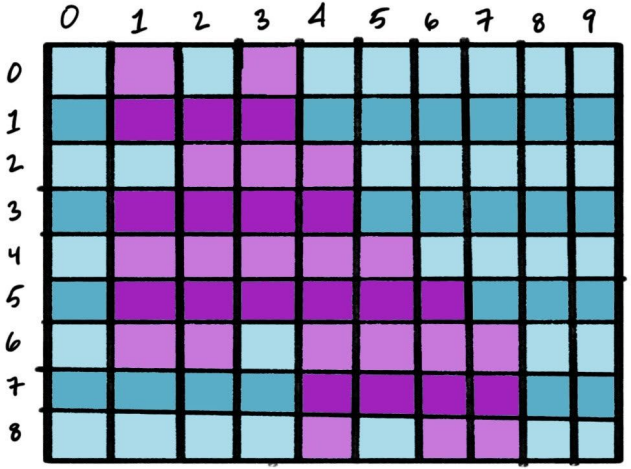
- Traditional Forward-Backward can be done using a [2 * Q] matrix, by storing the data row-wise.
- For Cloud Search, data must be held antidiagonal-wise in order to process them sequentially. This requires [3 * (Q + T)] memory. We require three rows because the diagonal dependency is two anti-diagonals back.
- Antidiagonal indices can be tracked where the sum of the coordinates is equal to the index of its antidiagonal. This creates the example recursion.





Step 1: Cloud Search

- Cloud Search stores the resulting search space as edgebounds, which describes each contiguous block of data on each given row or anti-diagonal.
- For each row, an row-wise edgebound array of length 10 is allocated.
- After each anti-diagonal is computed and pruned, the antidiag-wise edgebound is integrated on the fly into the existing row-wise edgebound.

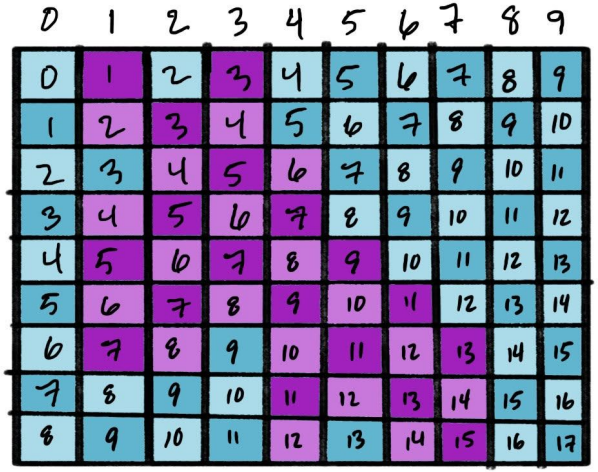


ROW-WISE

0: {1,2}, {3,4}
1: {1,4}
2: {2,5}
3: {1,5}
4: {1,6}
5: {1,7}
6: {1,3}, {4,8}
7: {4,8}
8: {4,5}, {6,8}

Step 1: Cloud Search

- Cloud Search stores the resulting search space as edgebounds, which describes each contiguous block of data on each given row or anti-diagonal.
- For each row, an row-wise edgebound array of length 10 is allocated.
- After each anti-diagonal is computed and pruned, the antidiag-wise edgebound is integrated on the fly into the existing row-wise edgebound.

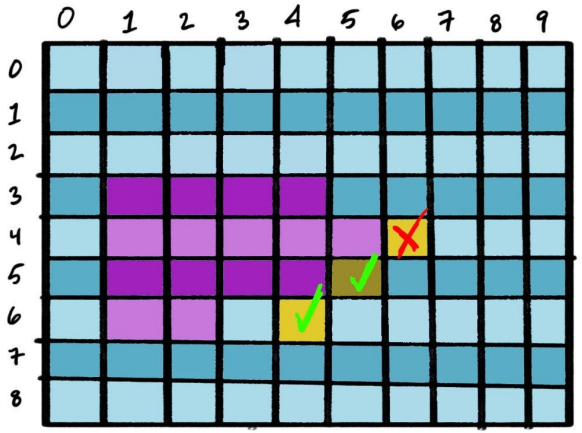


ANTI-DIAG

1: {1,2}
2: {1,2}
3: {2,4}
4: {1,4}
5: {1,4}
6: {1,5}
7: {1,5}
8: {2,5}
9: {4,6}
10: {4,6}
11: {4,7}
12: {4,7}
13: {6,8}
14: {6,8}
15: {7,8}

Step 1: Cloud Search

- Cloud Search stores the resulting search space as edgebounds, which describes each contiguous block of data on each given row or anti-diagonal.
- For each row, an row-wise edgebound array of length 10 is allocated.
- After each anti-diagonal is computed and pruned, the antidiag-wise edgebound is integrated on the fly into the existing row-wise edgebound.

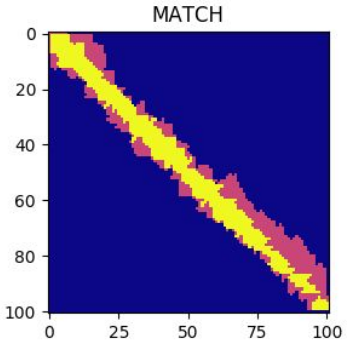


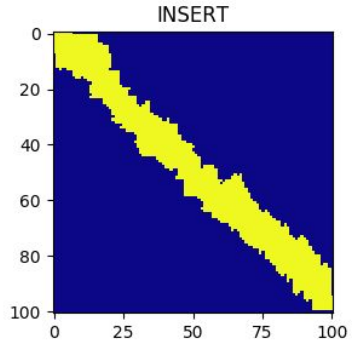
ROW-WISE

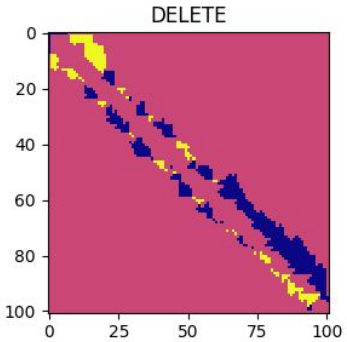
3: {1, 5}
4: {1, 6} ✗
5: {1, 5} + 5 ⇒ {1, 6}
6: {1, 3} + 4 ⇒ {1, 3}, {4, 5}

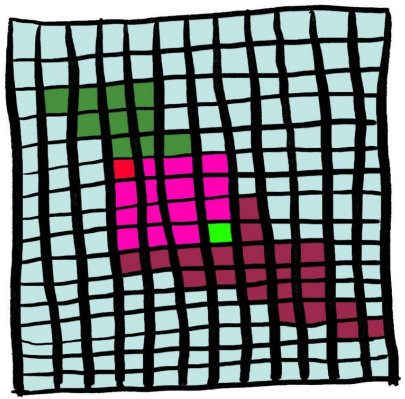
Step 1: Cloud Search

- We perform this method in the Forward and Backward direction.
- We then take the union of the cells computed during the forward and backward pass.
- Union is formed by performing mergesort on the Backward and Forward ordered Edgebound lists, then merging adjacent pairs which overlap.
- Once we have this unioned search space, we perform a bounded version of the traditional Forward-Backward algorithm.





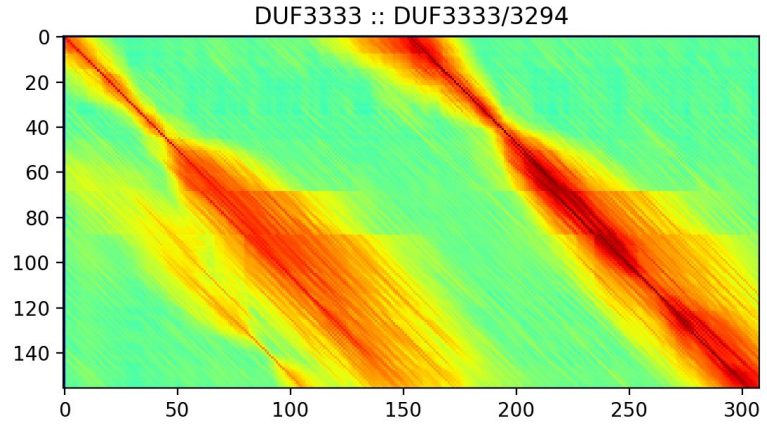


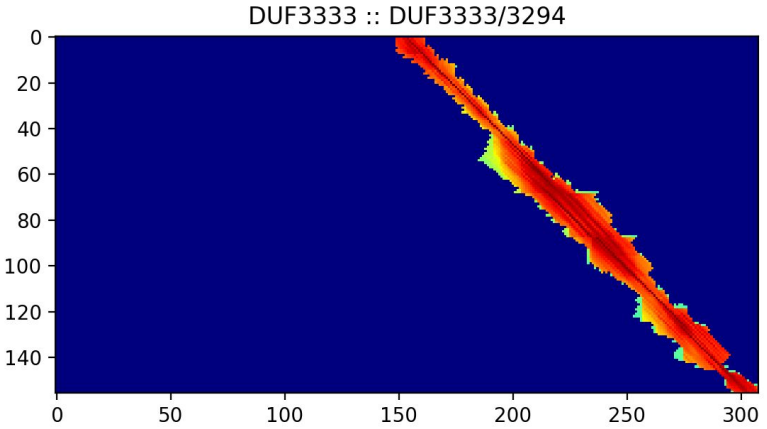


Step 1: Cloud Search

- If we find the proper parameters, a huge portion of the search space can be pruned (down from quadratic to approximately linear) with minimal cost to accuracy.
- This search space tends to scale approximately linearly with the window size of the alignment.

- Example: DUFF3333 v DUF3333/3294
  - Scores
    - Forward: 23.517
    - Bound Forward: 23.505
  - Cells
    - Total Cells: 48048
    - Cloud Cells: 3078
    - Computed 0.064061 of Total Cells

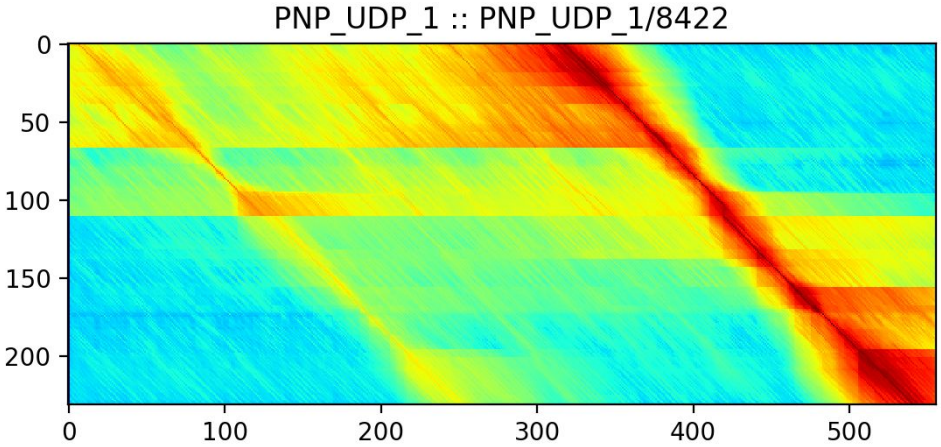


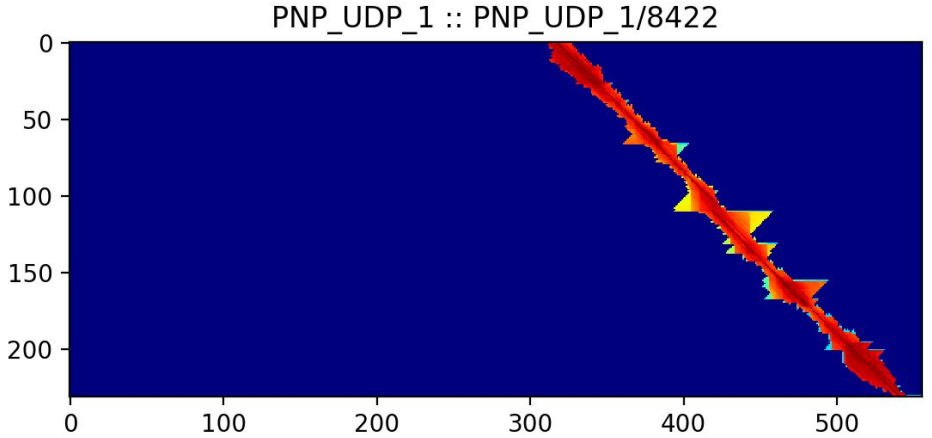


Step 1: Cloud Search

- If we find the proper parameters, a huge portion of the search space can be pruned (down from quadratic to approximately linear) with minimal cost to accuracy.
- This search space tends to scale approximately linearly with the window size of the alignment.

- Example: PNP_UDP_1 v PNP_UDP_1/8422
  - Score
    - Forward: 58.476
    - Bound Forward: 58.464
  - Cells
    - Total Cells: 128436
    - Cloud Cells: 5102
    - Computed 0.039724 of Total Cells

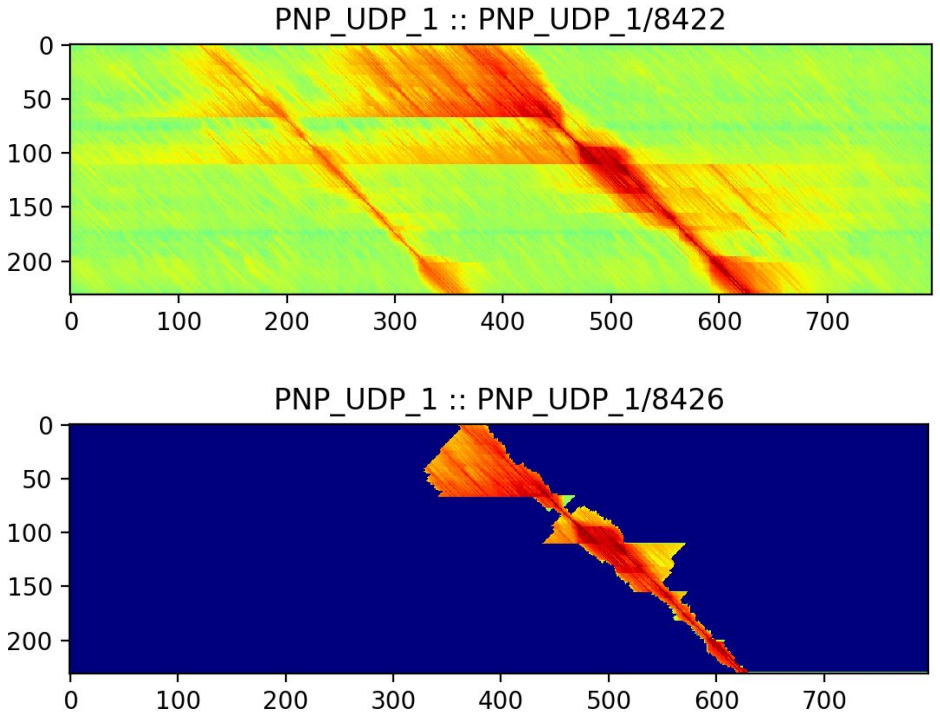




Step 1: Cloud Search

- If we find the proper parameters, a huge portion of the search space can be pruned (down from quadratic to approximately linear) with minimal cost to accuracy.
- This search space tends to scale approximately linearly with the window size of the alignment.

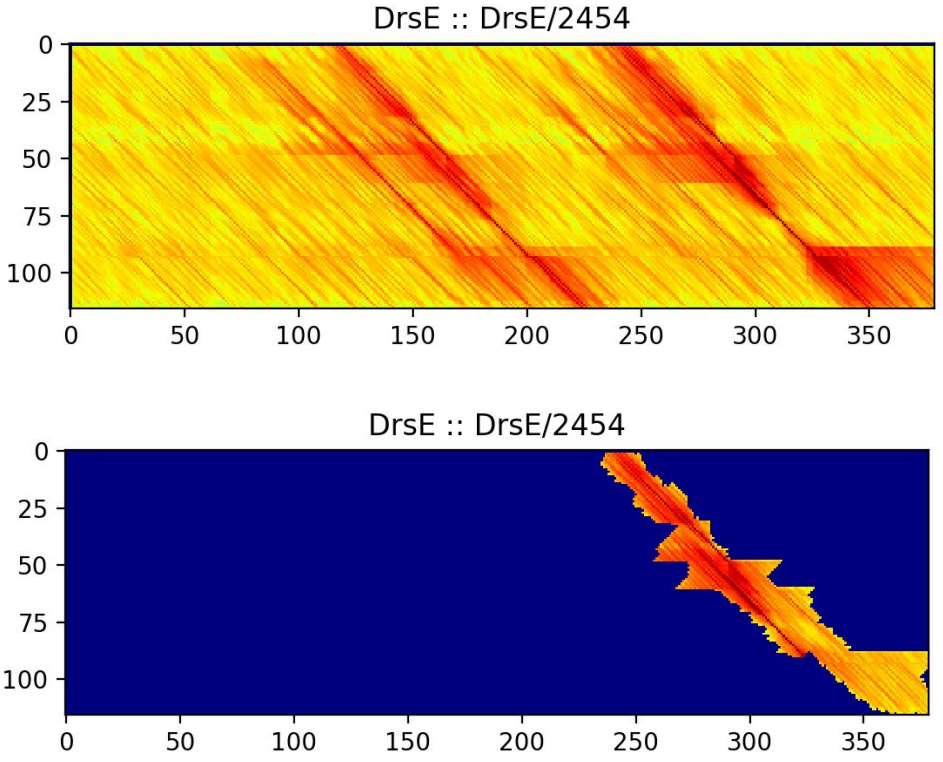
- Example: PNP_UDP_1 v PNP_UDP_1/8426
  - Score
    - Forward: 14.382
    - Bound Forward: 14.376
  - Cells
    - Total Cells: 184107
    - Cloud Cells: 11703
    - Computed 0.063566 of Total Cells



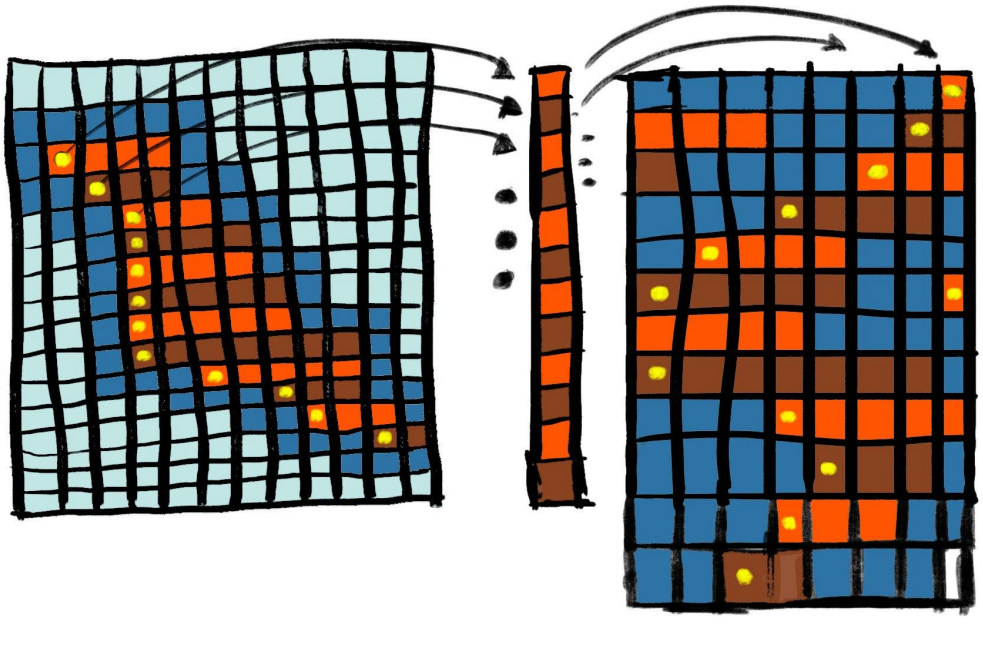
Step 1: Cloud Search

- If we find the proper parameters, a huge portion of the search space can be pruned (down from quadratic to approximately linear) with minimal cost to accuracy.
- This search space tends to scale approximately linearly with the window size of the alignment.

- Example: DrsE v.q DrsE/2454
  - Score
    - Forward: 1.339
    - Bound Forward: -1.215
  - Cells
    - Total Cells: 43964
    - Cloud Cells: 3712
    - Computed 0.084433 of Total Cells

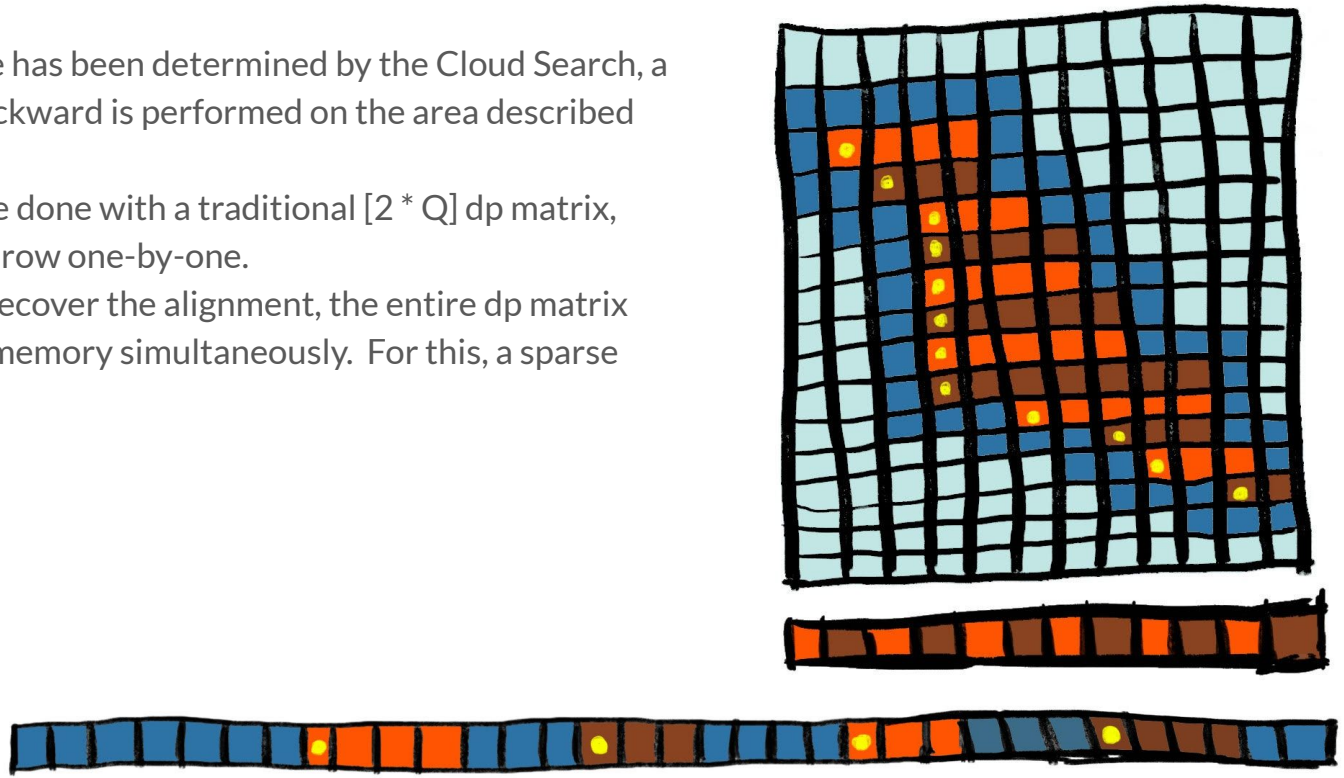


Step 2: Bounded Forward-Backward



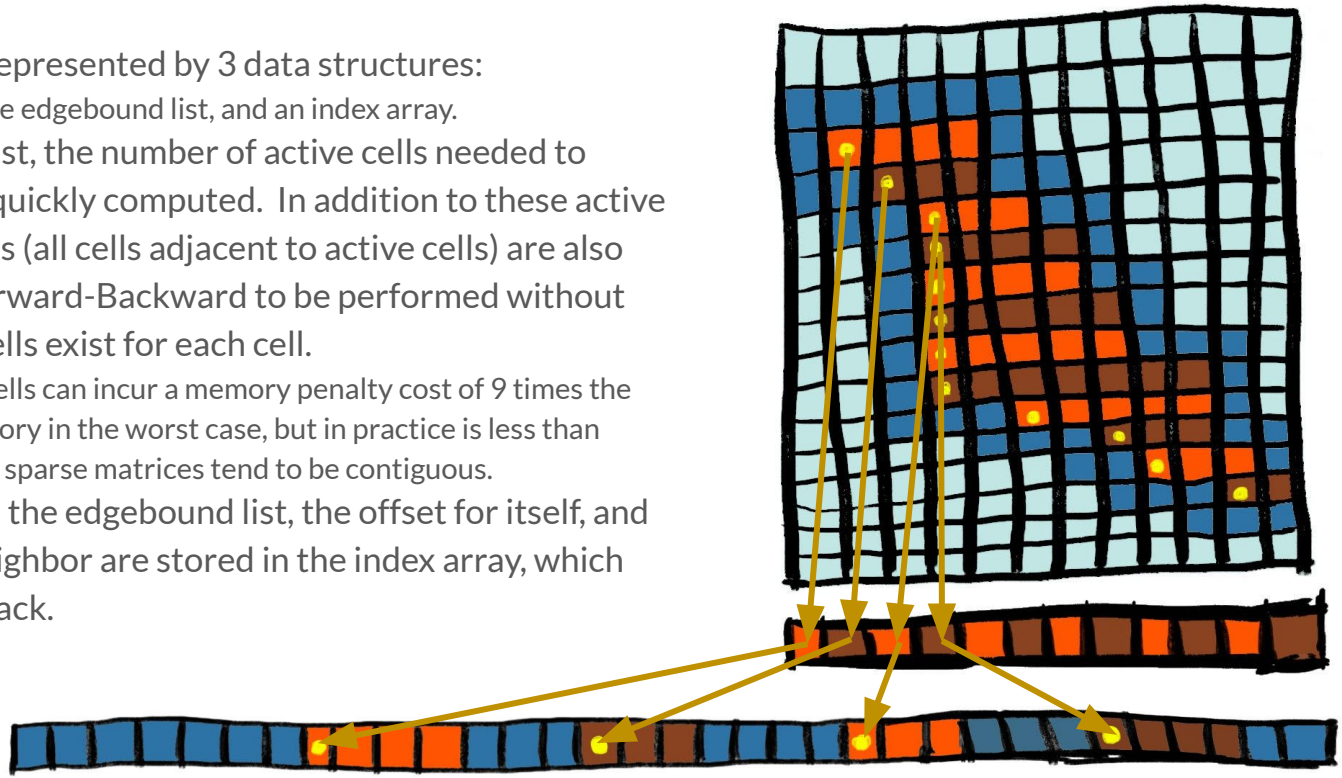
Step 2: Bound Forward-Backward

- Once the search space has been determined by the Cloud Search, a Bounded Forward-Backward is performed on the area described by the Edgebound list.
- Normally, this could be done with a traditional [2 * Q] dp matrix, which computed each row one-by-one.
- However, in order to recover the alignment, the entire dp matrix needs to be stored in memory simultaneously. For this, a sparse matrix is used.



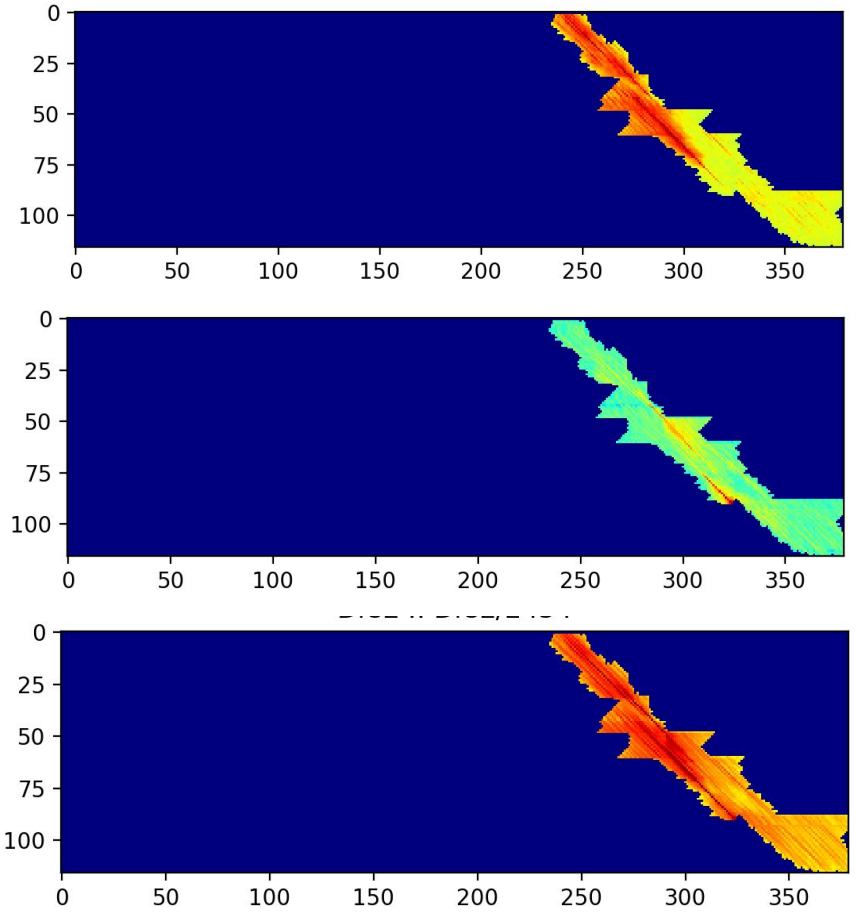
Step 2: Bound Forward-Backward

- The Sparse Matrix is represented by 3 data structures:
  - a flat data array, the edgebound list, and an index array.
- With the edgebound list, the number of active cells needed to store the data can be quickly computed. In addition to these active cells, all perimeter cells (all cells adjacent to active cells) are also added. This allows Forward-Backward to be performed without checking if adjacent cells exist for each cell.
  - These perimeter cells can incur a memory penalty cost of 9 times the active matrix memory in the worst case, but in practice is less than double since these sparse matrices tend to be contiguous.
- Then for each range in the edgebound list, the offset for itself, and its north and south neighbor are stored in the index array, which allows for quick lookback.



Step 3: Alignment Recovery

- Once we have completed the Forward-Backward passes, all that is left is to do alignment recovery.
- This is performed as is done the same as it is done in the HMMER pipeline.
- The posterior probability is computed by multiplying the forward and backward probabilities.
- Finally, bias correction is computed for the length of the sequence.



Results

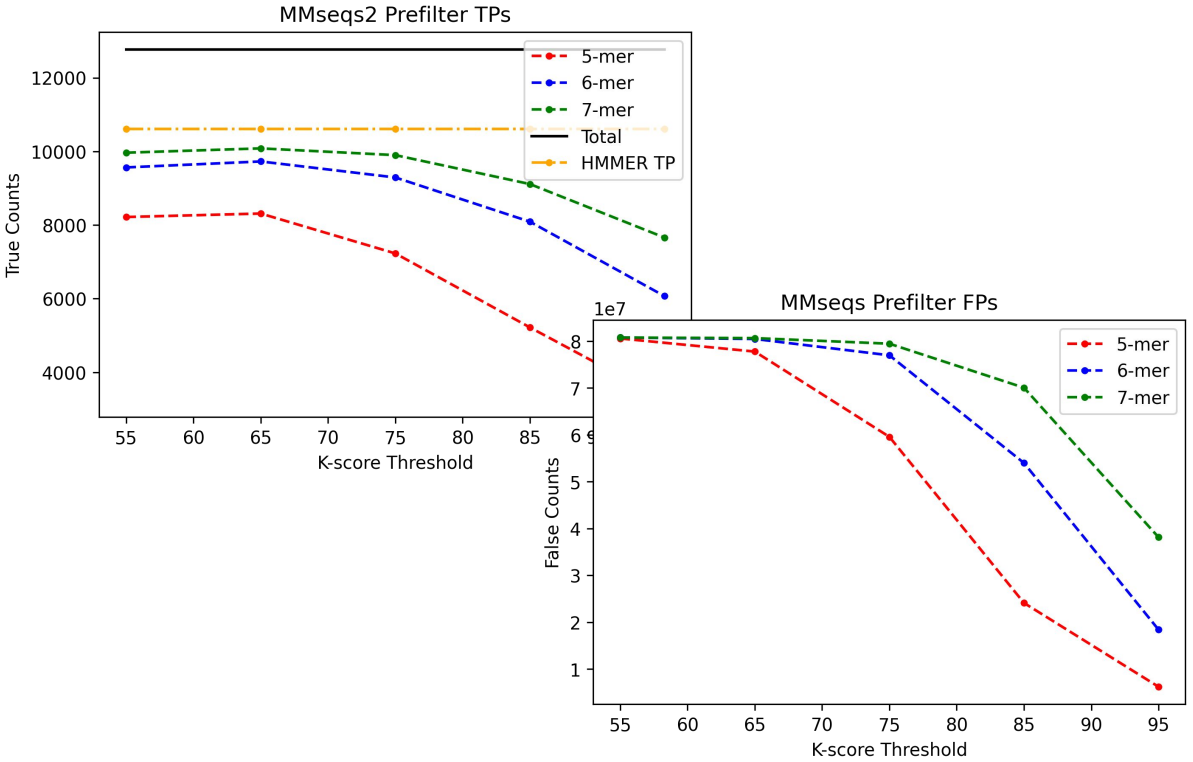
Datasets Used

There were two primary datasets used in the benchmarks:

1. Profmark++ Dataset (fewer FPs)
    - 2,000,000 decoys, 17088 reals
    - 3003 queries
    - Decoys contain shuffled real sequences
2. Profmark Dataset (fewer FPs)
    - 200,000 decoys, 17088 reals
    - 3003 queries
    - Decoys contain shuffled real sequences
3. MMseqs Dataset
    -

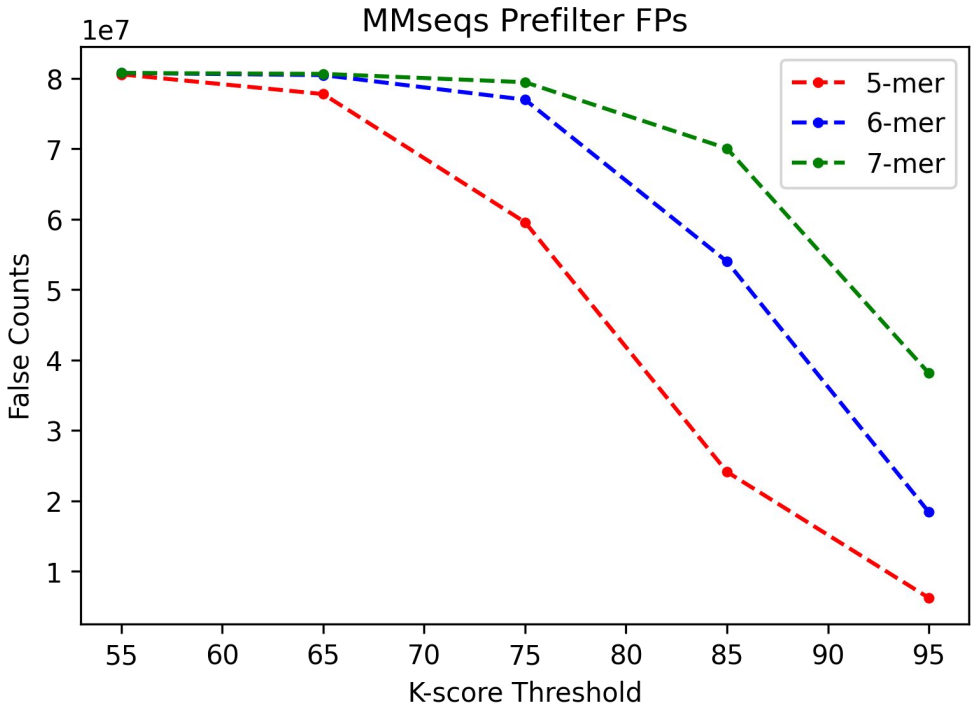
MMseqs Parameter Selection

- MMseqs was run using different parameters for its double K-mer prefilter:
  - Length of K-mer
  - Score threshold for K-mer
- The default K-mer length of 7 was found to produce the best results.
- The score threshold cutoff was a compromise between efficiency and accuracy. For most best overall results, 65 was the point where nearly where we ceased to see any further improvement. However, this meant allowing many more FPs through the prefilter.
- For most tests, 75 was settled on for the threshold, which was tested alongside the mmseqs fast and sensitive params, 95 and 80.



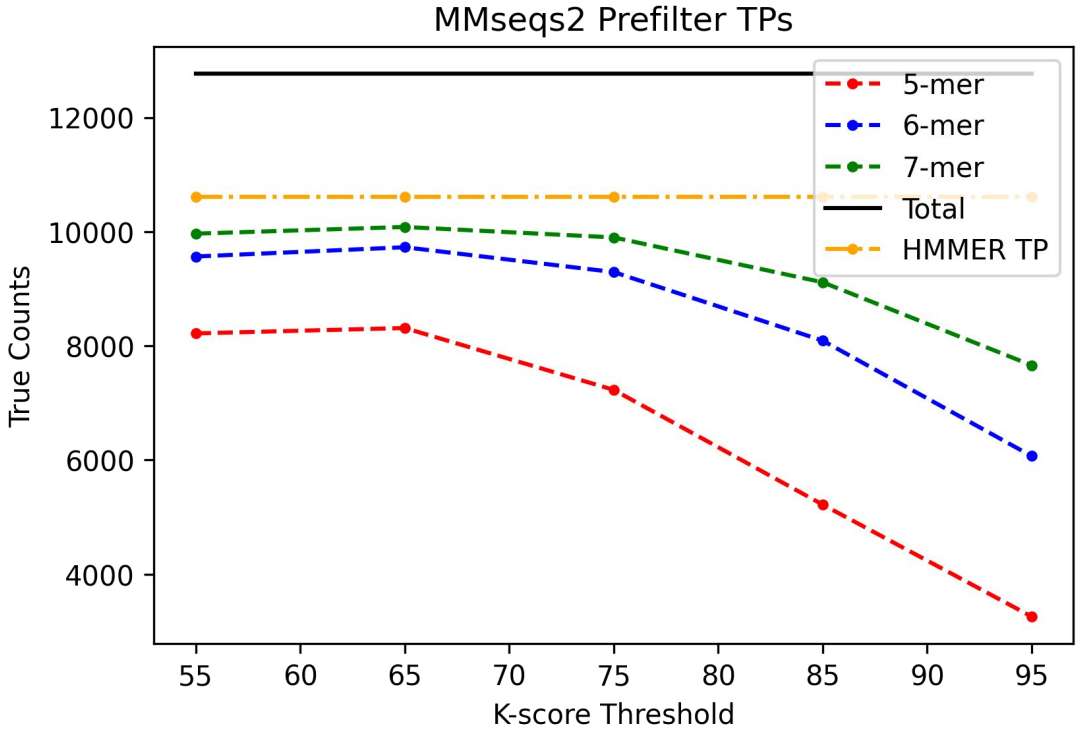
MMseqs Parameter Selection

- MMseqs was run using different parameters for its double K-mer prefilter:
  - Length of K-mer
  - Score threshold for K-mer
- The default K-mer length of 7 was found to produce the best results.
- The score threshold cutoff was a compromise between efficiency and accuracy. For most best overall results, 65 was the point where nearly where we ceased to see any further improvement. However, this meant allowing many more FPs through the prefilter.
- For most tests, 75 was settled on for the threshold, which was tested alongside the mmseqs fast and sensitive params, 95 and 80.



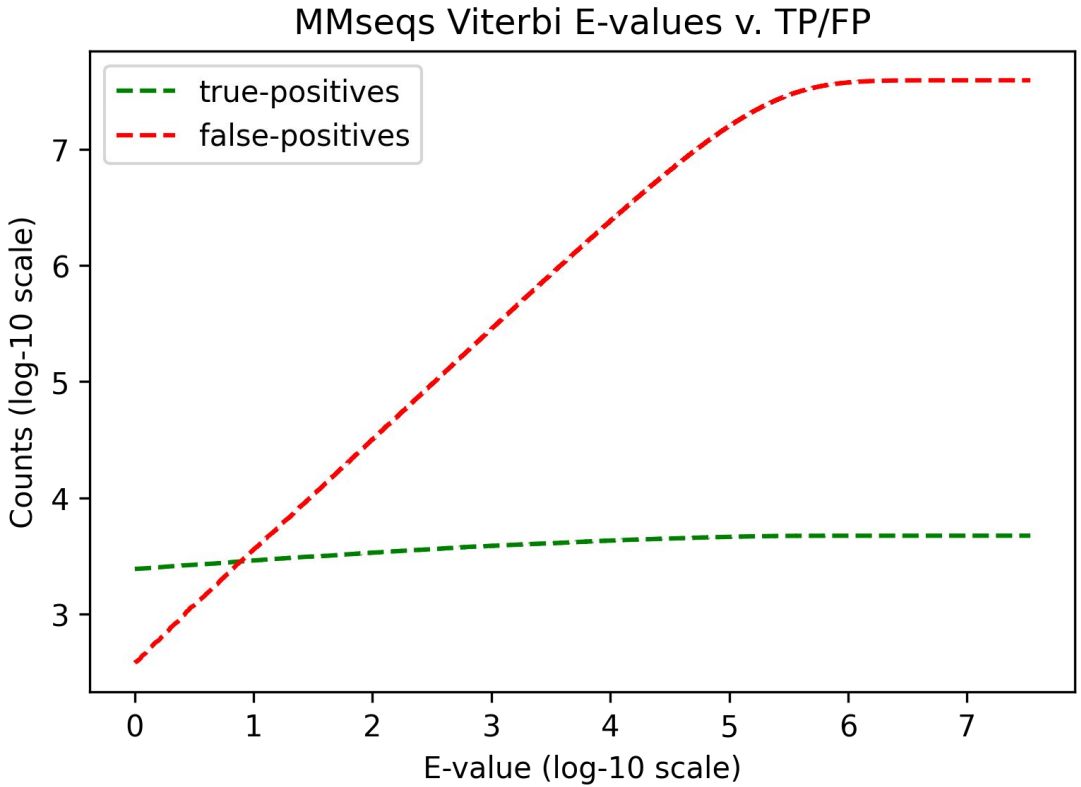
MMseqs Parameter Selection

- MMseqs was run using different parameters for its double K-mer prefilter:
  - Length of K-mer
  - Score threshold for K-mer
- The default K-mer length of 7 was found to produce the best results.
- The score threshold cutoff was a compromise between efficiency and accuracy. For most best overall results, 65 was the point where nearly where we ceased to see any further improvement. However, this meant allowing many more FPs through the prefilter.
- For most tests, 75 was settled on for the threshold, which was tested alongside the mmseqs fast and sensitive params, 95 and 80.



MMseqs Parameter Selection

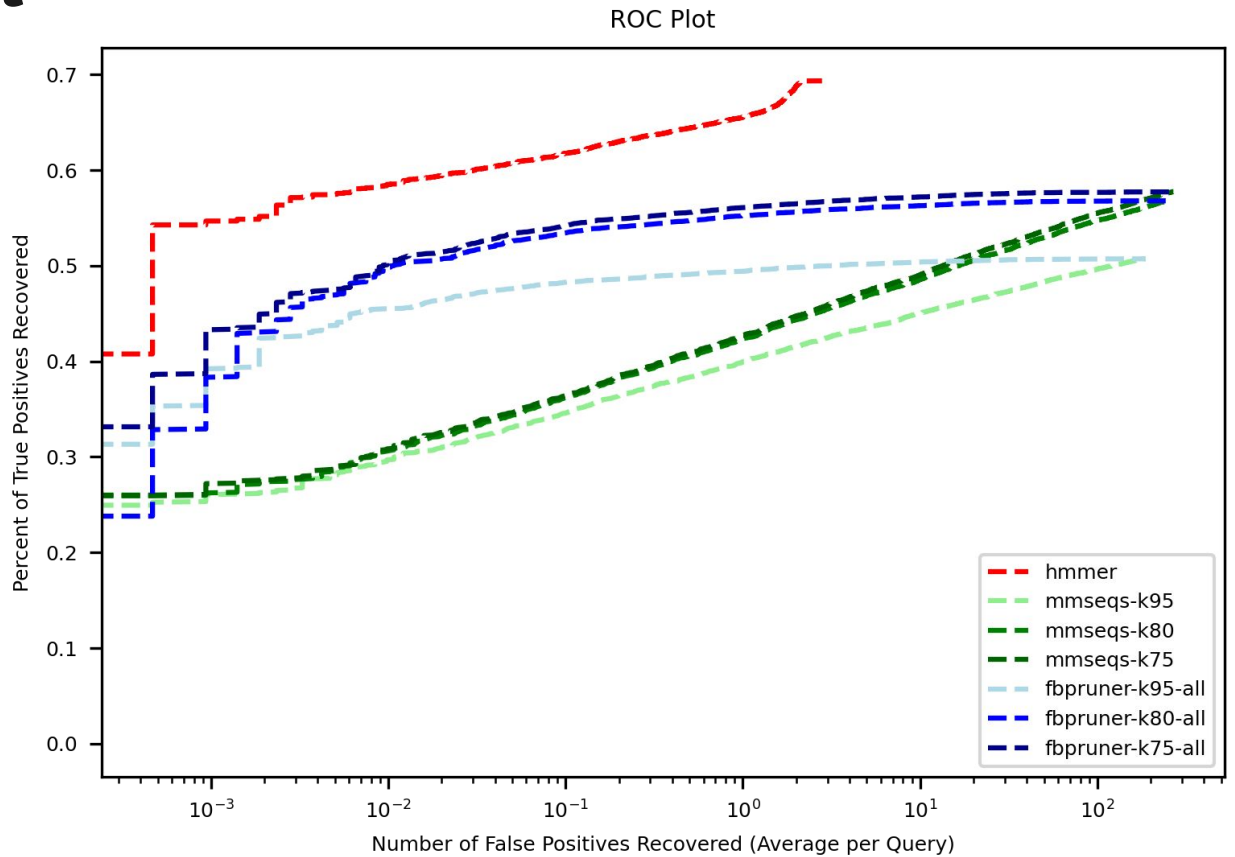
- For MMORE, the MMseqs Viterbi filter was run piped with two parameters:
  - No cutoff used.
  - P-value = 0.001 used, same as HMMER's Viterbi cutoff threshold.



Profmark: ROC Plot

- Profmark was ran against several different MMseqs prefilter parameters.
- Green: Results from MMseqs before being passed onto the Pruned-FB.
- Blue: Results from MMORE.
- Red: Results from HMMER.

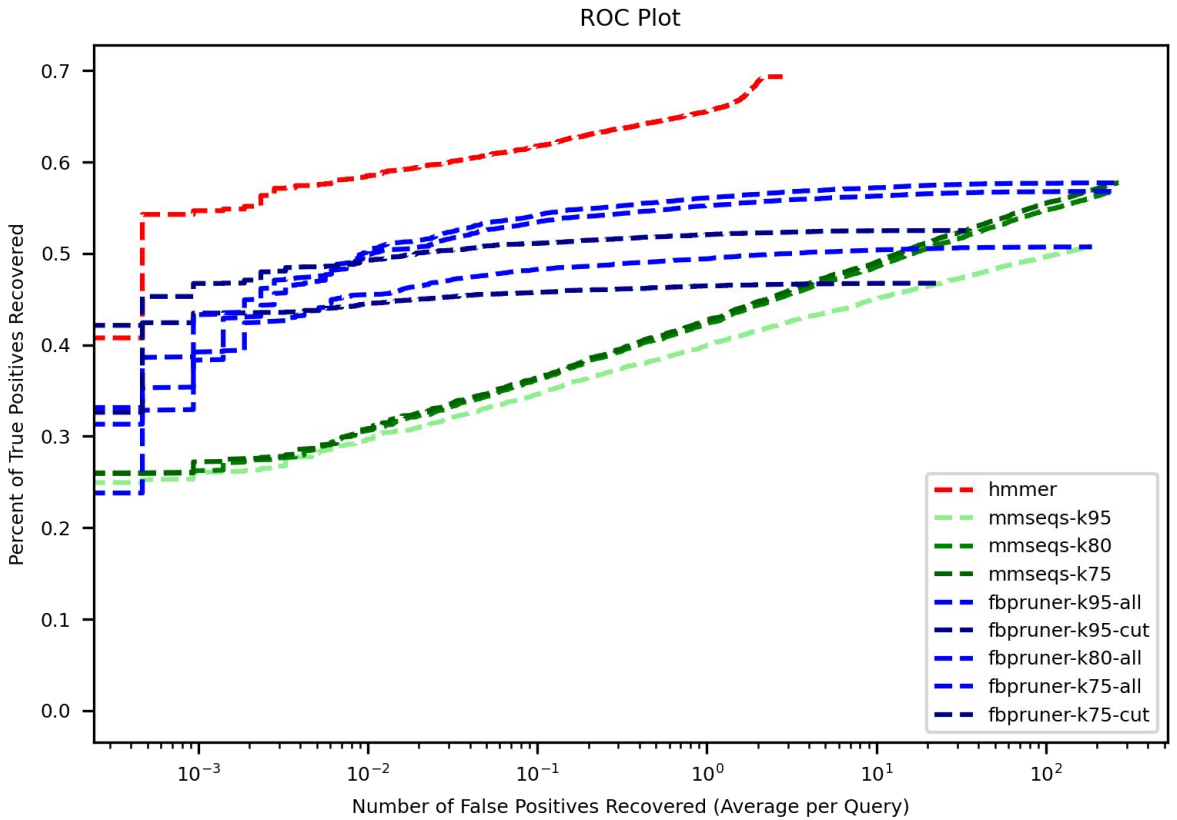
- Note: These results currently do not account for bias correction.



Profmark: ROC Plot

- Profmark was ran against several different MMseqs prefilter parameters.
- Green: Results from MMseqs before being passed onto the Pruned-FB.
- Blue: Results from MMORE with no Viterbi cutoff.
- Dark Blue: Results from MMORE with no cutoff.
- Red: Results from HMMER.

- Note: These results currently do not account for bias correction.

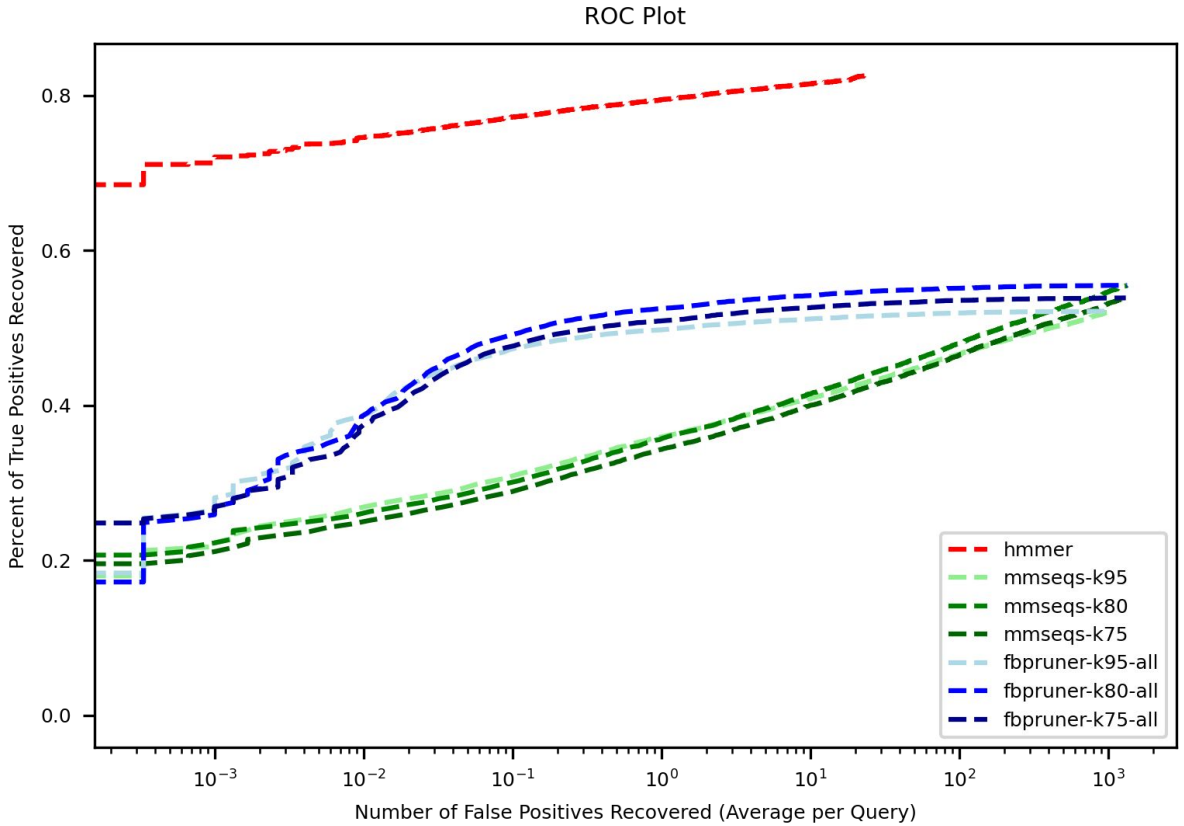


Profmark+: ROC Plot w/ No cutoff

- Profmark was ran against several different MMseqs prefilter parameters (normal, sensitive, plus).

- Green: Results from MMseqs before being passed onto the Pruned-FB.
- Blue: Results from MMORE with no Viterbi cutoff.
- Red: Results from HMMER.

- Note: These results currently do not account for bias correction.

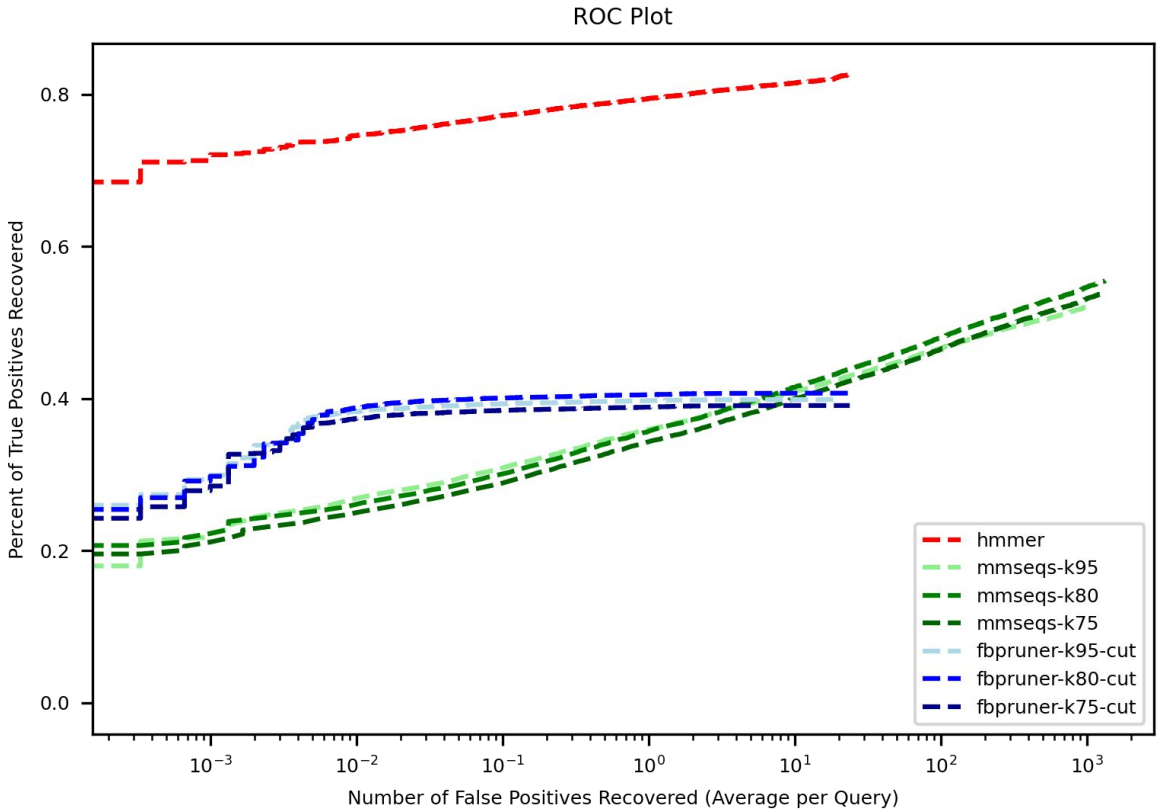


Profmark+: ROC Plot w/ cutoff

- Profmark was ran against several different MMseqs prefilter parameters (normal, sensitive, plus).

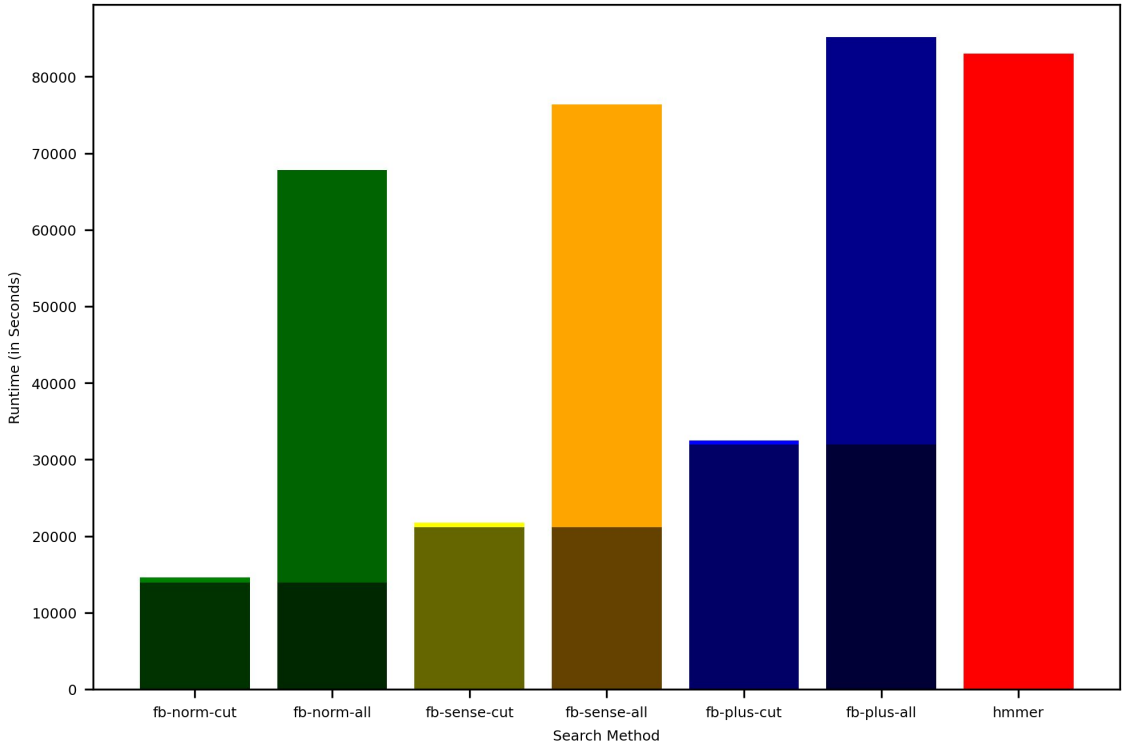
- Green: Results from MMseqs before being passed onto the Pruned-FB.
- Blue: Results from MMORE with Viterbi cutoff.
- Red: Results from HMMER.

- Note: These results currently do not account for bias correction.



Profmark+: Time Benchmark

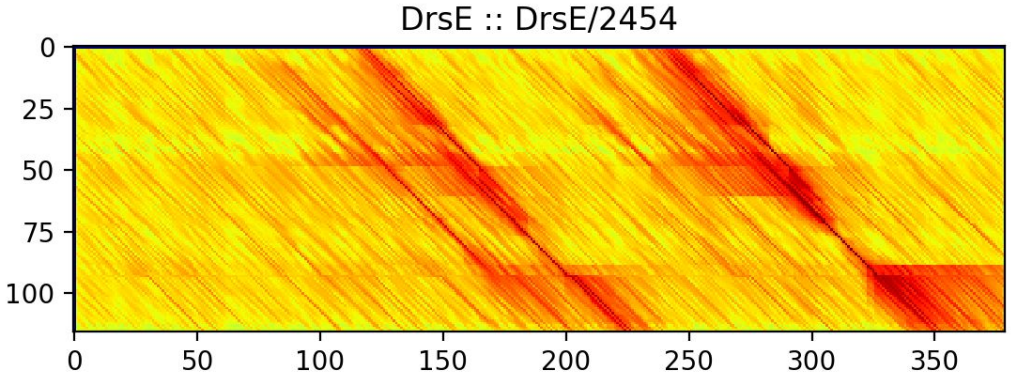
- MMORE was run with 3 different parameters for the MMseqs prefilter: norm=95, sensitive=80, plus=75.
- MMORE was run with 2 different parameters for MMseqs Viterbi: all= No cutoff, cut= P=0.001



Conclusion

Future Improvements

- Speed/Accuraccy: Parameter selection has not been optimized. There is probably a better MMseqs Viterbi cutoff threshold between P=0.001 and none.
- Speed: There is some potential for SIMD vectorization.
- Accuracy: Using jump states to allow for multiple viterbi alignments seeds. (MMseqs currently only supplies the highest scoring Viterbi alignment.)
- Speed: Potentially using cutoffs at the Cloud Search and Bound Forward-Backward steps.



Thanks to...

- Questions?

Conclusion

- Questions?