Applied Genomics - Lecture 12

Arun Das

Working with Genomic Data

- Genomic data is BIG
  - A single human genome is 3GB, and data structures built on this may be larger
- Need our algorithms and approaches to scale well

There are two sides to this problem - working with large data structures, and reducing large data into smaller, manageable parts.

Efficiently using large data structures

Efficiently using large data structures

- Often have large data structures that we need to query/search to find what we need
- Can we search these efficiently?

Suffix Arrays

- Given a string **s** and a pattern **p**, find all occurrences where **p** occurs as a substring of **s**
  - One of the most common problems in genomics
- Substring search is a core component of many read and whole genome aligners
  - “Seed” part of seed-and-extend
- Suffix arrays are one of the most widely used approaches to accelerate this problem
  - Fast, space-efficient, powerful

```
S = "ACCATGATGG"
p = "CAT"
```

0	ACCATGATGG
3	ATGATGG
6	ATGG
2	CATGATGG
1	CCATGATGG
9	G
5	GATGG
8	GG
4	TGATGG
7	TGG

Searching a Suffix Array

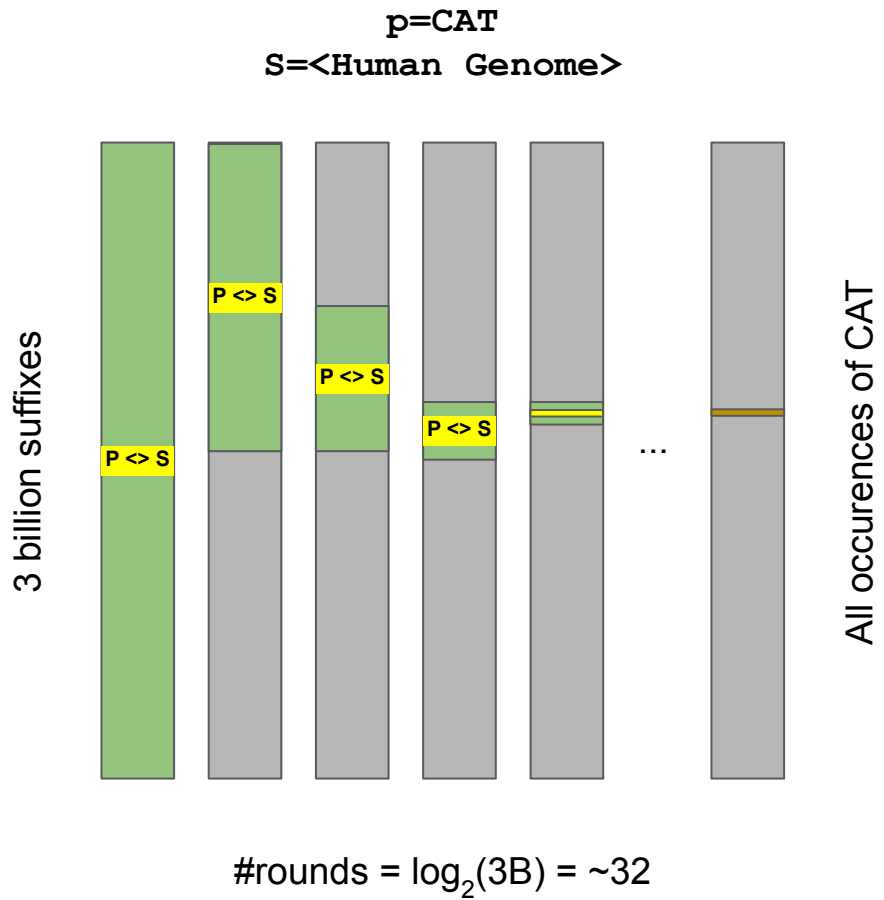
- By indexing all possible suffixes, the suffix array indexes all substrings of S
  - Each occurrence of a pattern **p** in a string **s** corresponds to a prefix (the beginning) of some suffix of **s**
  - Solution consists of a contiguous range of rows, each row corresponding to a specific offset of the original string

```
S = "ACCATGATGG"
p = "AT"
```

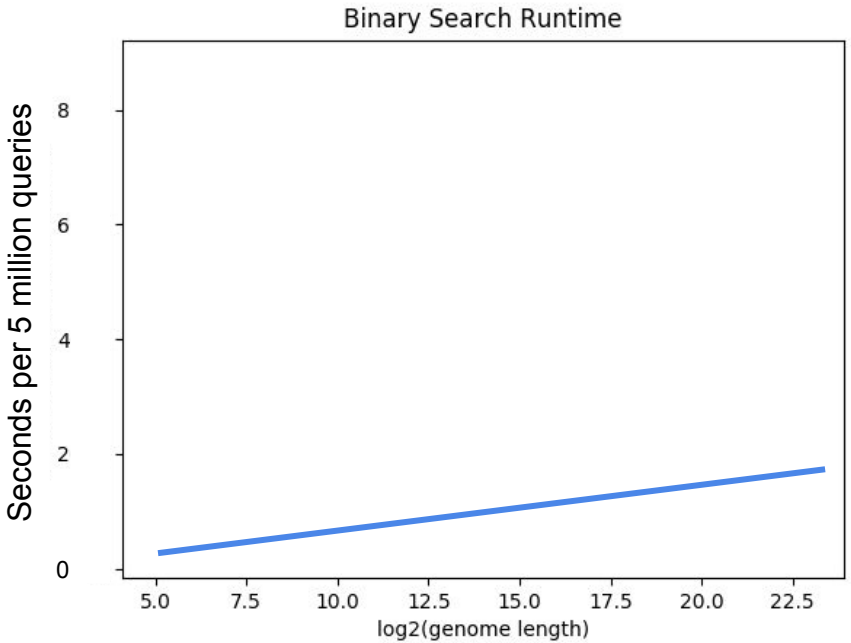
0	ACCATGATGG
3	**AT**GATGG
6	**AT**GG
2	CATGATGG
1	CCATGATGG
...	

Traditional Suffix Array Search

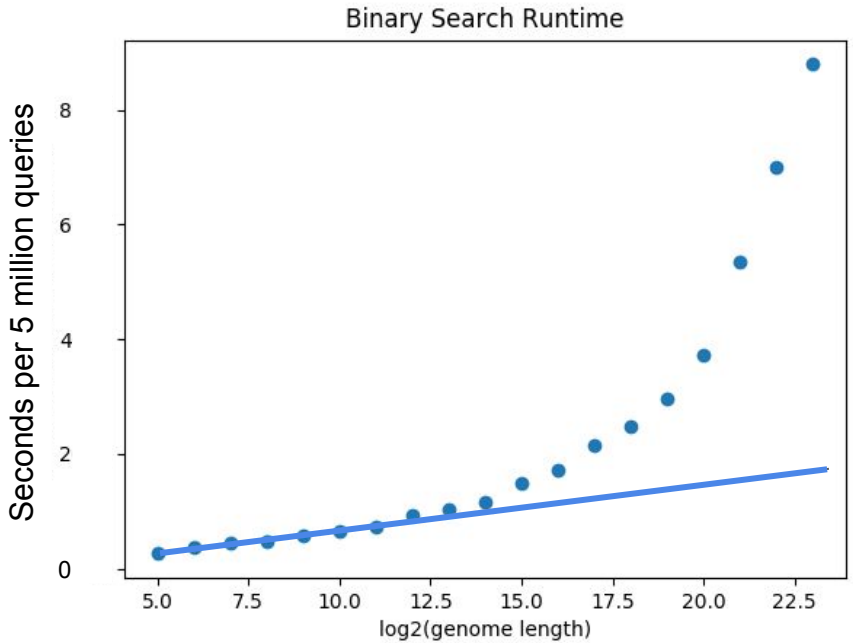
- Binary search can be used to find the correct row
- Takes log(# of rows) iterations to find



Performance



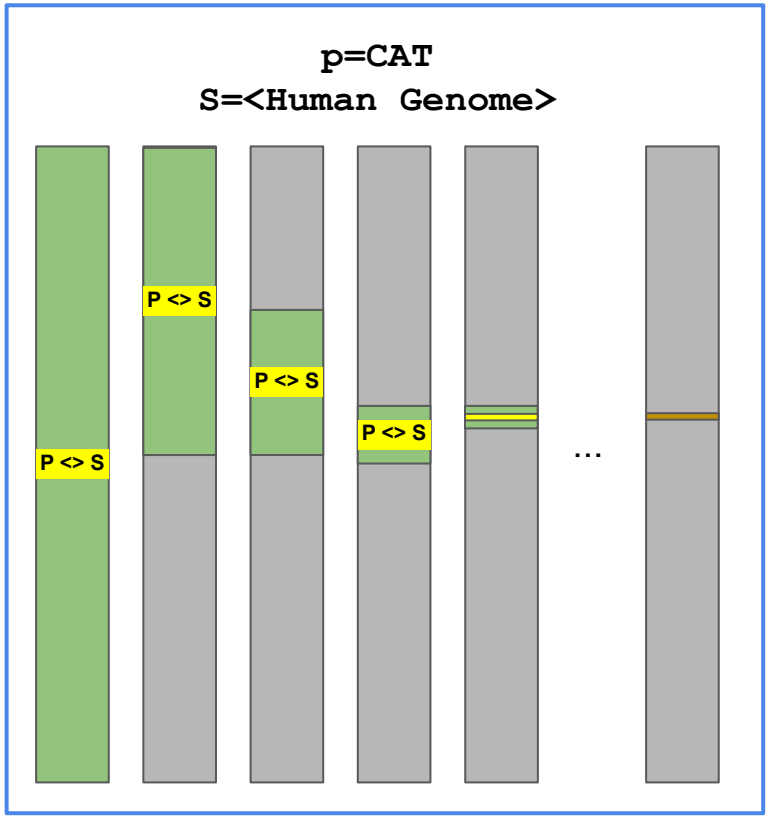
In theory, searching should scale linearly with $\log_2$ of the genome size

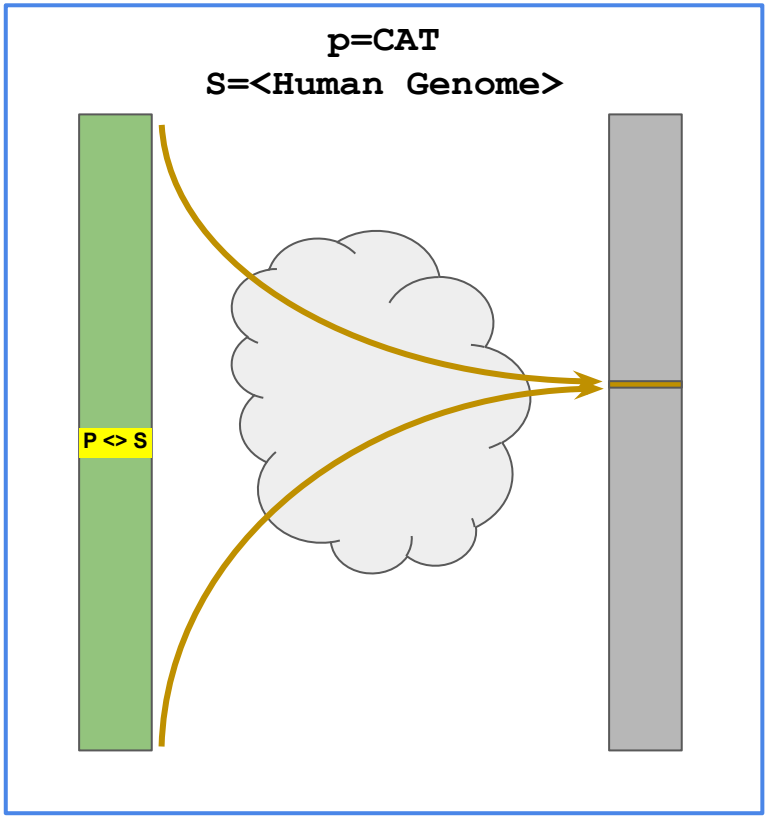


In practice, searching is much slower for large genome sizes

Binary search suffers from poor locality causing many slow lookups in main memory

Improving on binary search

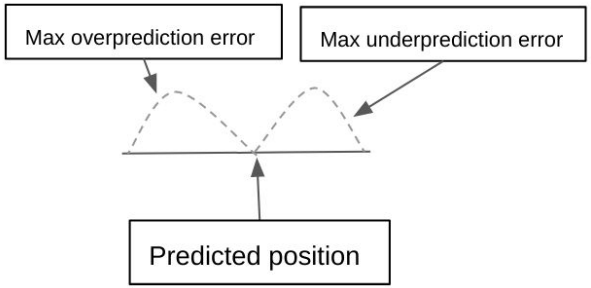




What if we could quickly guess/predict the correct rows? → **Learned Data Models**

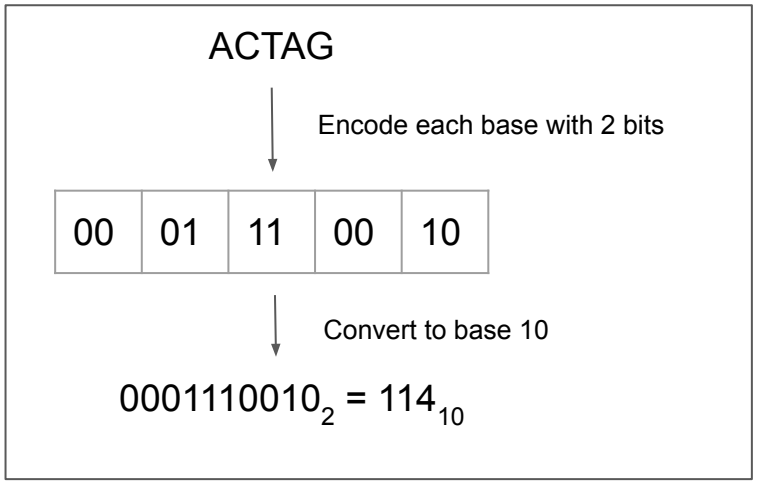
Learned Index Structures

- Researchers at Google using neural networks to augment classical data structures such as B-Trees, HashMaps, and Bloom Filters
  - Train network to predict position of a data point in the structure given its value
- Compute the maximum error E = |predicted position - actual position| among all points in data structure.
  - Then, narrow search to within E of predicted value.



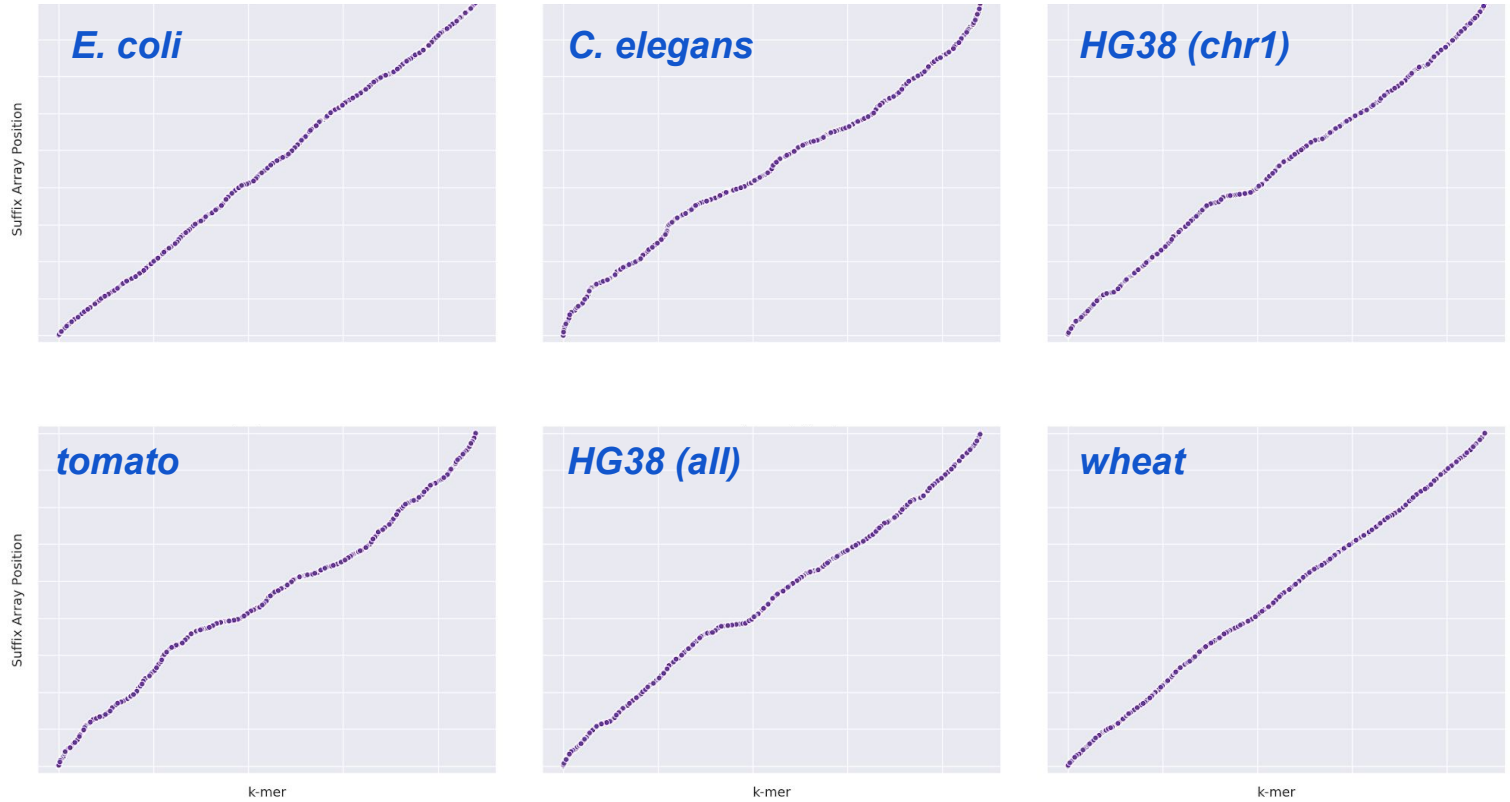
Kraska et. al. “The Case for Learned Index Structures”. SIGMOD 2018.

Suffix Array Search as a Prediction Task

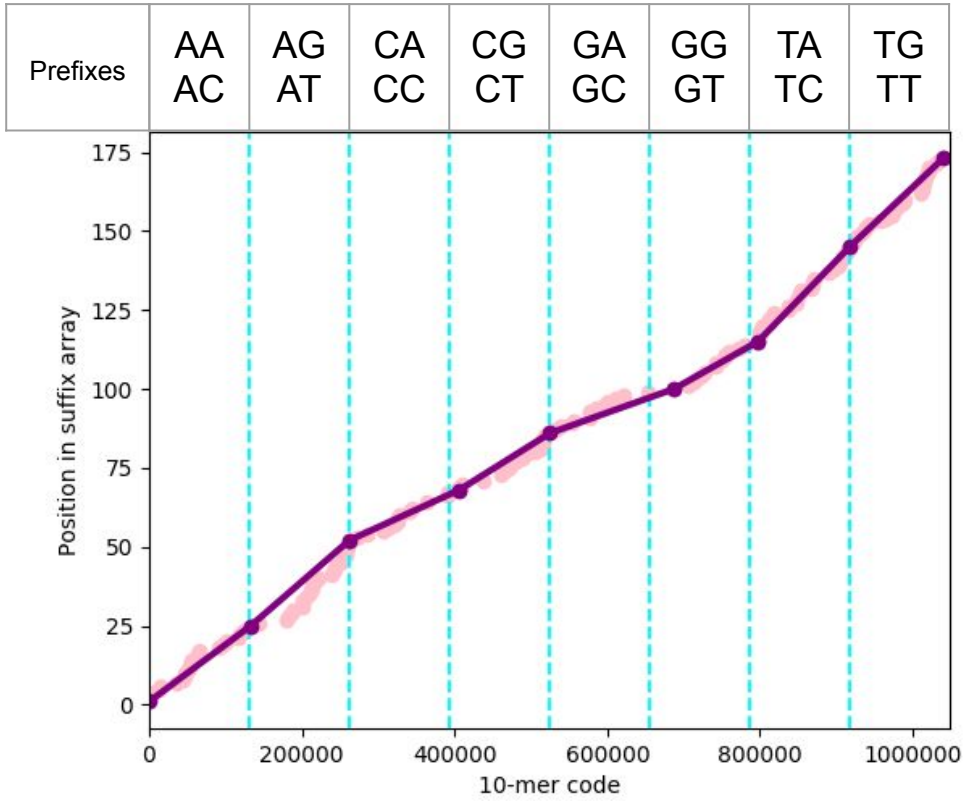


Goal: Given a query string and a suffix array, predict the suffix array position where the suffix begins with that query string

Suffix Array Search as a Prediction Task



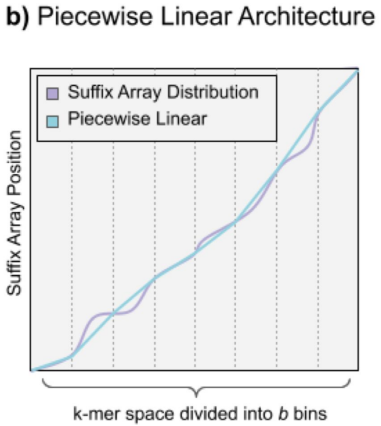
Piecewise Linear (PWL) Approach



- Divide k-mer space into buckets, and fit a linear function to each
  - Results in a piecewise linear function over the genome
  - Predicts k-mer position within a small range which is then searched
- Much smaller range to be searched
- Results in fast and accurate suffix array querying!

Piecewise Linear (PWL) Approach

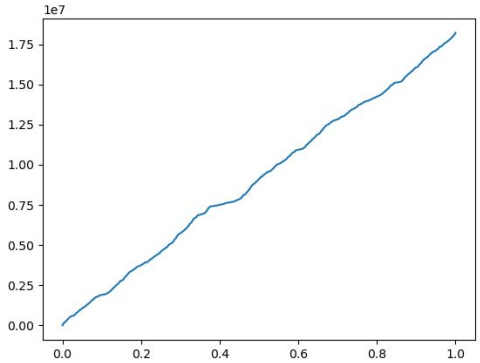
- Piecewise Linear approach is EXTREMELY compact
- Each linear function just needs two integer values - a gradient and an intercept
- Can divide the genome into a very large number of pieces, as there is not much overhead

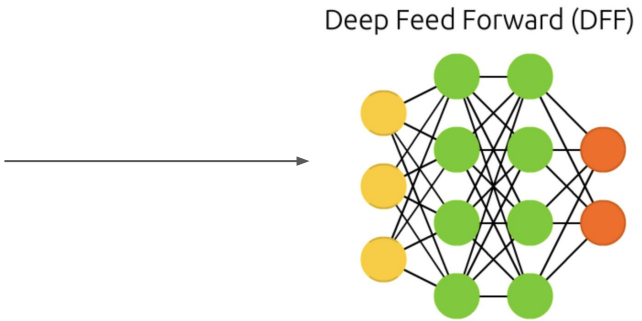


Neural Nets

- Piecewise Linear involves fitting a simple linear function to each segment
- What if we used a more complex function? What if we fit this function over a larger segment of the genome?

Neural Nets, as universal approximators*, can be used for this

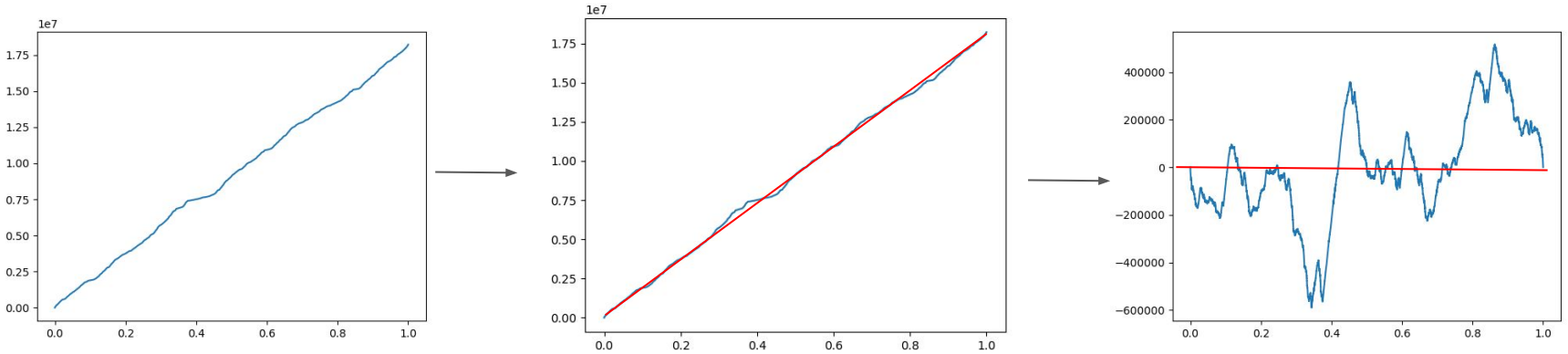




*Cybenko. “Approximation by Superpositions of a Sigmoidal Function”. MCSS 1989.

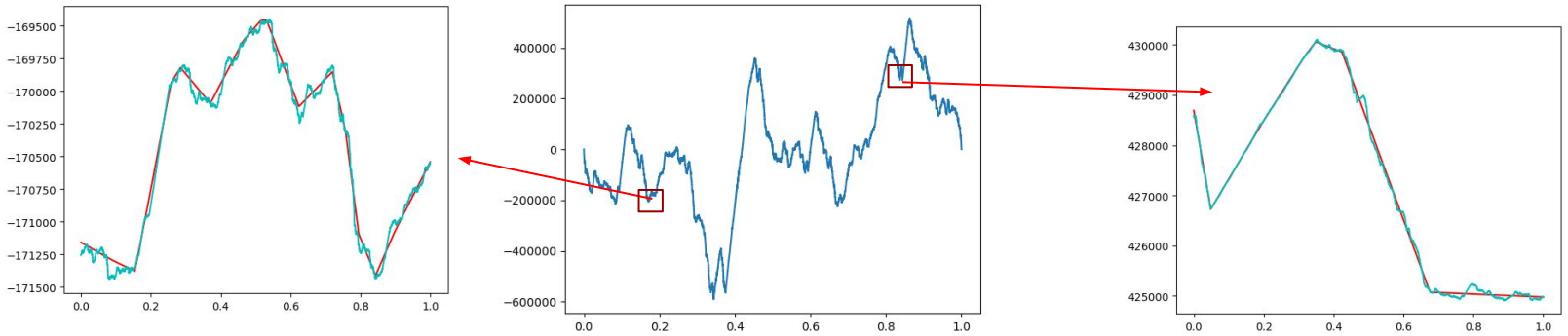
Pre-processing and Training

- “Detrending”
  - Measure of how far from the best fit line each suffix truly is
- Allows us to see how complex the approximation needs to be



Pre-processing and Training

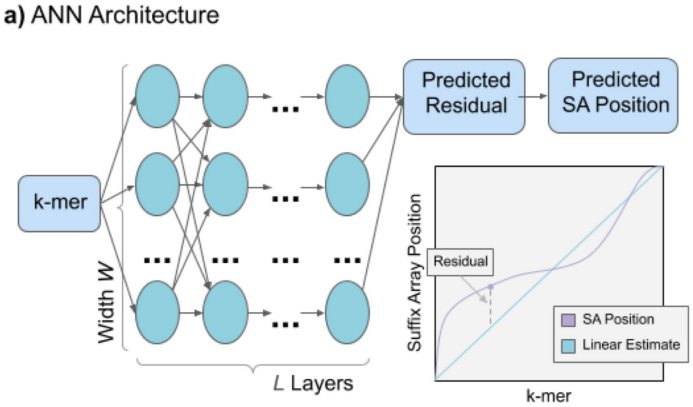
- Models are then trained on this “residual data”
- For chr22, we trained 1000 models to learn the function
  - Below, we see the plot for all of chr22, and then how accurately the models for two chunks fit the function



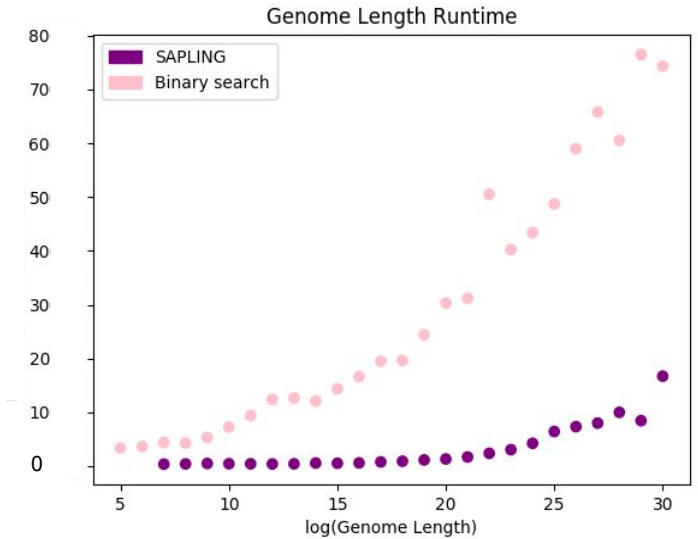
Chr22

Neural Net Approach

- Employ a “chunk”-based approach
  - Similar to buckets in piecewise linear function
- Divide the suffix array into ~1000 chunks, fit a relatively small model (one layer of varying width) to each of the chunks
- For querying, find the appropriate model for the k-mer, and make the prediction for it.



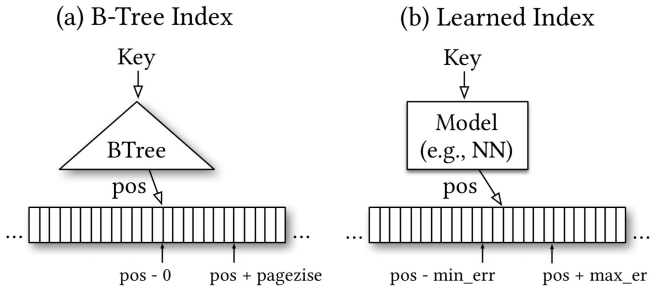
Performance and Results



- Both approaches perform well!
- Accuracy is good, reduces the range that needs to be searched for a particular row.
- Neural Net has more overhead (model vs a simple function), but needs fewer partitions.
- PWL has very low mean/median error, NN bounds max error better.

The Future of Learned Index Structures

- Learned index structures could be the future of querying and interacting with large data structures and databases
- Able to greatly reduce the search range when looking for a result, speeding up search
- As long as you can generate a pattern between position and the data, you can try to train a model to recognize and predict it!



Working with smaller representations

Working with smaller representations

- What if we could only use a part of the input data, but get accurate results?
- Want to generate a reduced representation of the input data, and process that
- Want results on the reduced representation to accurately reflect results on the original data

We want this reduced representation to be an accurate representation of the original data!

Using reduced representations in read classification

- The biggest read classification approaches in use today are mostly index- or alignment based
- These have large space/time overheads
- What if we could classify with a much smaller footprint?

Improved metagenomic analysis with Kraken 2

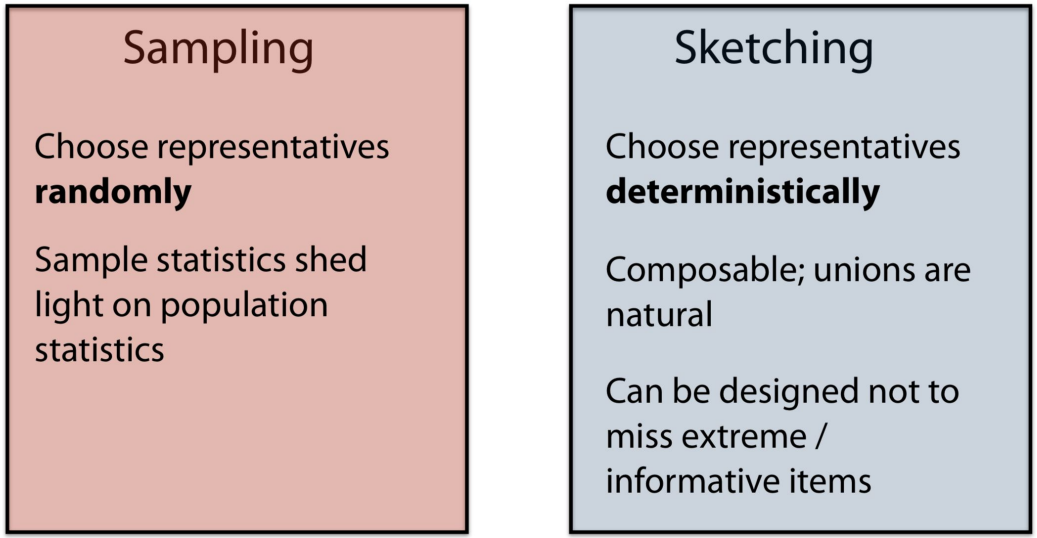
Derrick E. Wood, Jennifer Lu & Ben Langmead

Minimap2: pairwise alignment for nucleotide sequences FREE

Heng Li

Sketching and Sampling

- Both methods of reducing size of input data
- Generate reduced representations that we can compare and use



Ben Langmead

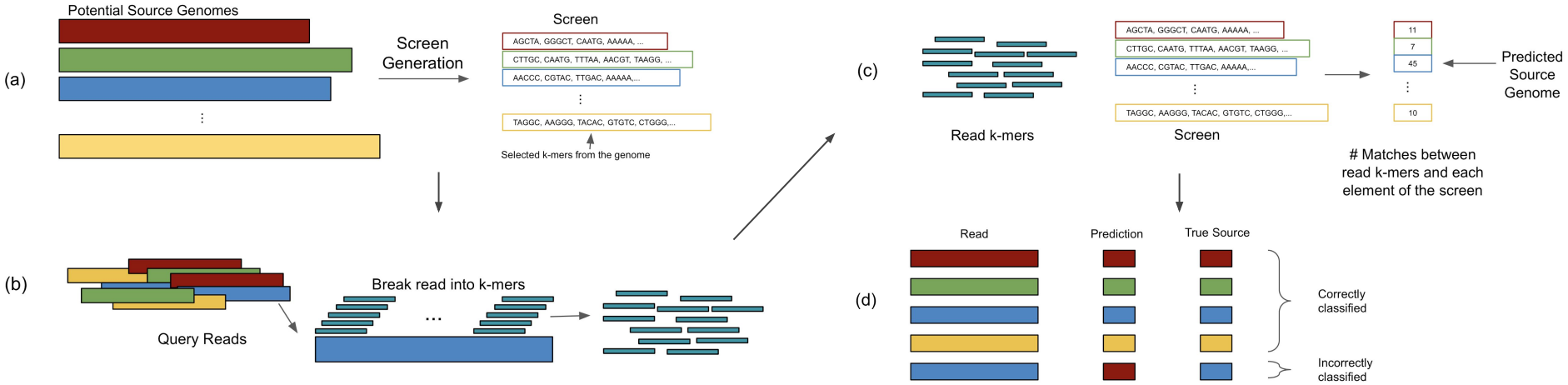
Using Sketching and Sampling in Read Classification

- Read Classification involves
  - Set of reads to classify
  - Set of genomes the reads could come from
- We want to avoid comparing the entirety of every read to the entirety of every genome
- Instead, we can shrink one of these, saving time and space

Compare reduced representation of the genomes to the reads!

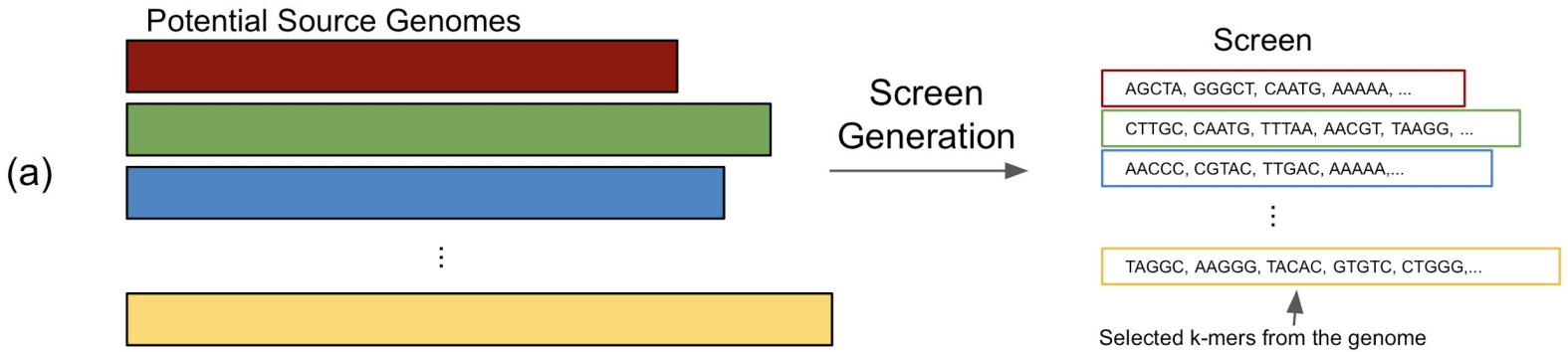
Read Classification - Overview

- Generating a reduced representation of potential source genomes (a), stream in the reads to be classified (b)
- Compare these reduced representations (c), predict from there (d)

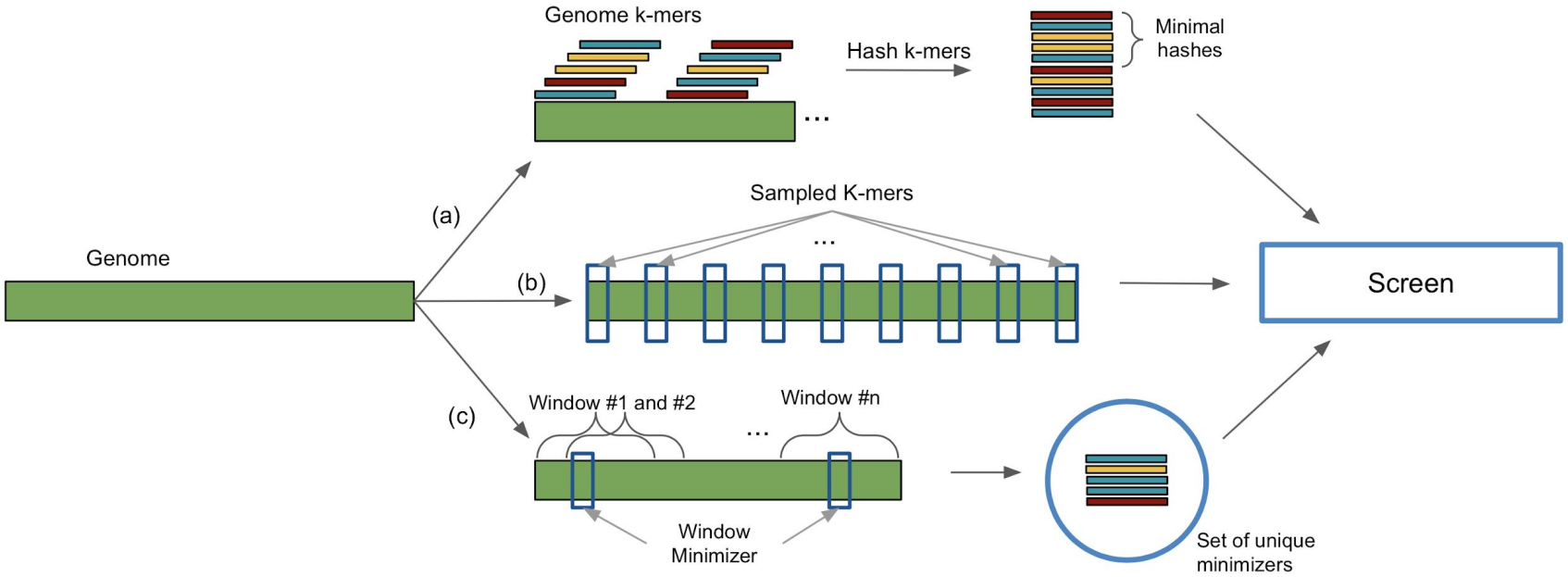


Screen Generation

- Screen = collection of reduced representations of potential source genomes
- If we sample/sketch the genomes carefully, the screen will be an accurate representation of the genomes



Screen Generation

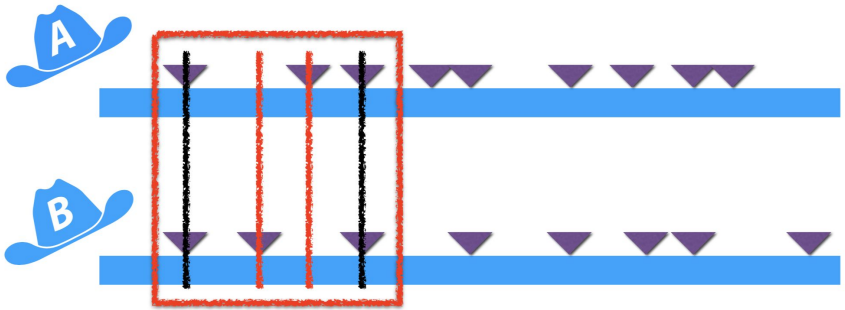


MinHash

- We focus on the MinHash sketch
- Hashes all subparts of a document, retains the minimal k hashes as a representation of the document
- Can compare MinHash sketches of two documents

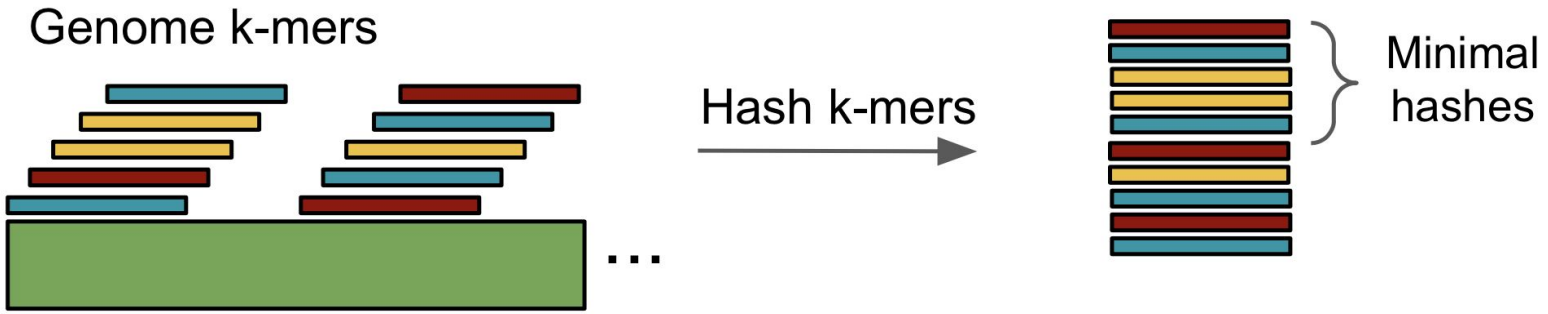
On the resemblance and containment of documents

Andrei Z. Broder
DIGITAL Systems Research Center
130 Lytton Avenue, Palo Alto, CA 94301, USA
broder@pa.dec.com



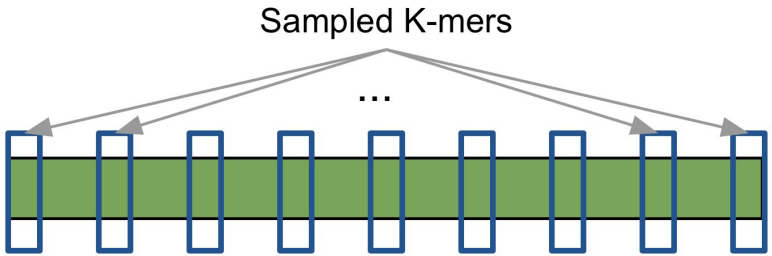
Screen Generation: MinHash

- Decompose genome into k-mers, retain the minimal k-mer hashes
- Essentially a random sampling of n k-mers from the genome
- Can potentially miss entire regions, though it is unlikely



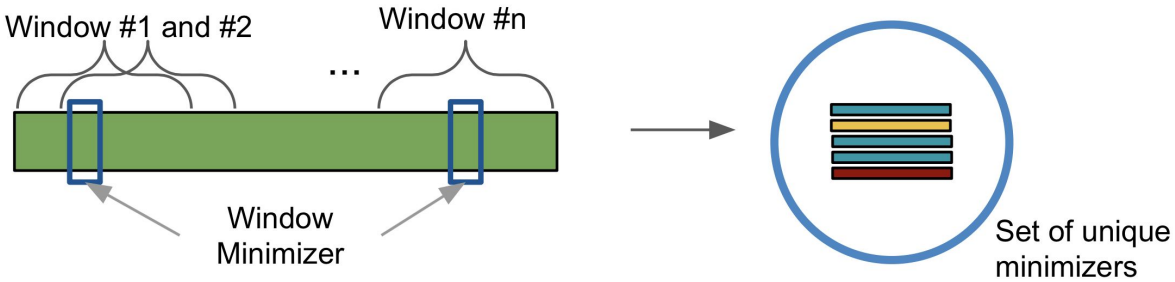
Screen Generation: Uniform Sampling

- Choosing a distance between samples, and then sample k-mers uniformly across the genome
- Ensures we have samples from across the genome, and ensures there is some overlap with every read



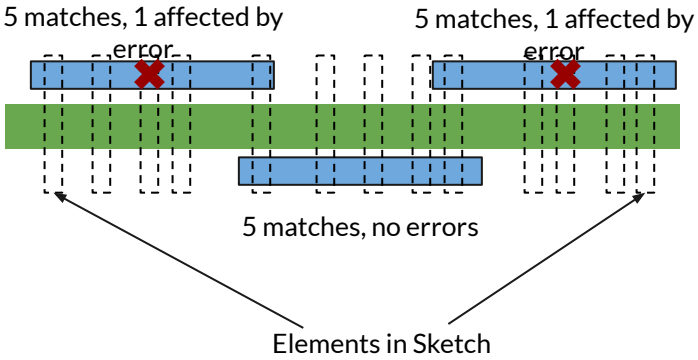
Screen Generation: Minimizers

- Sliding window of size n over the genome
- In each window, keep the minimal k-mer (i.e. k-mer with smallest hash)
  - This is the minimizer
- The set of unique minimizers is a representation of the genome

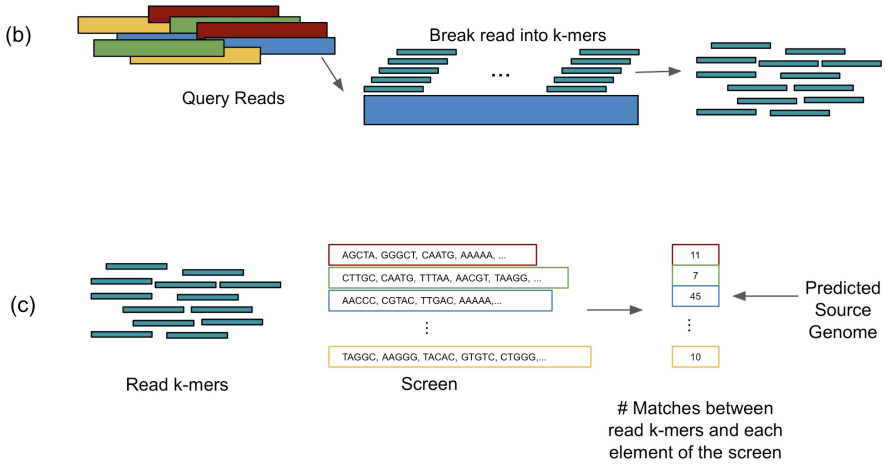


Screen Generation: Accounting for Error

- In our reduced representations, we want to account for the effect of error
- We oversample/”over-sketch” to account for the fact that errors can affect matches



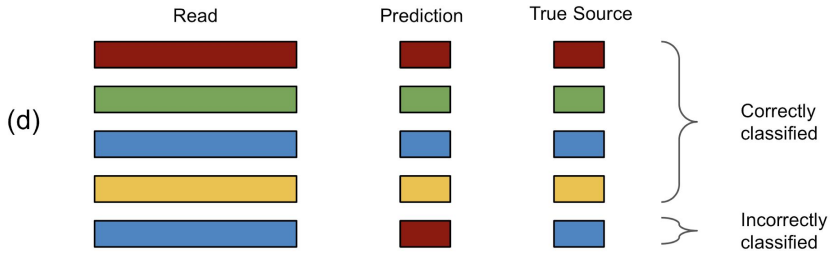
Read Screening



- Stream in input reads
- Read's k-mer list is then compared against sketch
- For each sketched genome, we count number of shared hashes between the read and that genome
- Genome with the highest number of matches is the prediction

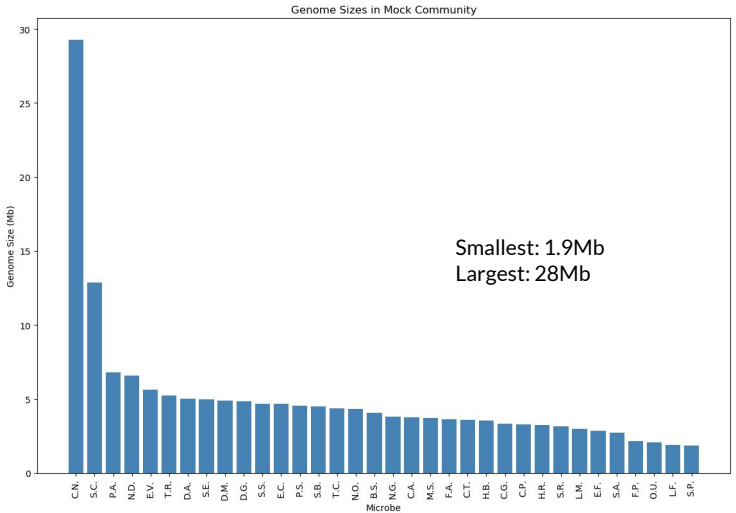
Read Screening

- We do this for every input read in a dataset
- When using simulated reads, we know the ground truth, so accuracy can be computed and reported
- For real datasets, we can use the output of Kraken/Minimap2 as the “ground truth”



Performance

- Using a microbial community of 34 microbes, totalling 170Mb of sequence
- Target of 30 matches and 10KB/1% error rate reads
- Screen representation is ~600K integers
- Reads are simulated from community, and streamed in, screened and classified
- 99.99% classification accuracy



ZYMO RESEARCH

Next generation sequencing data of a defined microbial mock community

Esther Singer, Bill Andreopoulos, Robert M. Bowers, Janey Lee, Shweta Deshpande, Jennifer Chiniquy, Doina Ciobanu, Hans-Peter Klenk, Matthew Zane, Christopher Daum, Alicia Clum, Jan-Fang Cheng, Alex Copeland & Tanja Woyke

Performance

- High Accuracy
- Misclassification between very similar organisms

	Screen Size	Correctly Classified	Incorrectly Classified
MinHash	623,862	165,859	29
Uniform	623,511	165,860	28
Minimizer	626,037	165,862	22

Next steps for this work

- Continued testing on larger and larger datasets
- A binary search tree based approach
  - Compare against less and less downsampled representations of the genomes

Where do we go from here?

- Developing faster and faster techniques to query and use large data structures
  - Learned index structures/data models are here to stay!
  - As the hardware/software for NNs gets better, these approaches keep getting faster
- Developing methods to sketch/sample input data and work with smaller versions

Could be an interesting topic for a course project!

Questions?