# SAPLING: Suffix Array Piecewise Linear INdex for Genomics

**Michael Kirsche**
**mkirsche@jhu.edu**

StringBio 2018

# Outline

- Substring Search Problem
- Caching and Learned Data Structures
- Methods
- Results
- Ongoing work

# Substring Search Problem

One of the most common problems in genomics:

***Given a string s and a pattern p,***

***Find all occurrences where p occurs as a substring of s***

Example:

`S = "ACCATGATGG"`

`p = "CAT"`

Substring search is a core component of many read and whole genome aligners including Bowtie, BWA, STAR, BLAST, MUMmer, and many others

- "Seed" portion of Seed-and-Extend algorithms

# Suffix Arrays

Several data structures have been proposed for accelerating the substring search problem including suffix trees, suffix arrays, BWT/FM-index, Hash tables, etc

***The suffix array is one of the most widely used approaches***

- The SA is the ***sorted*** list of the suffixes of a string stored implicitly as the index where each suffix begins
- ***Very fast:*** Faster than FM-Index using binary search-like algorithm
- ***Space Efficient***: Less compact than FM-index, although very practical on modern servers
- ***Powerful:*** Supports variable length queries

(Manber & Meyers, 1990)

S=CATTAGA

| |
|---|
| [6] A |
| [4] AGA |
| [1] ATTAGA |
| [0] CATTAGA |
| [5] GA |
| [3] TAGA |
| [2] TTAGA |

# Suffix Array Queries

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=CATGCGCATGCTAGCATCAT
p=CAT

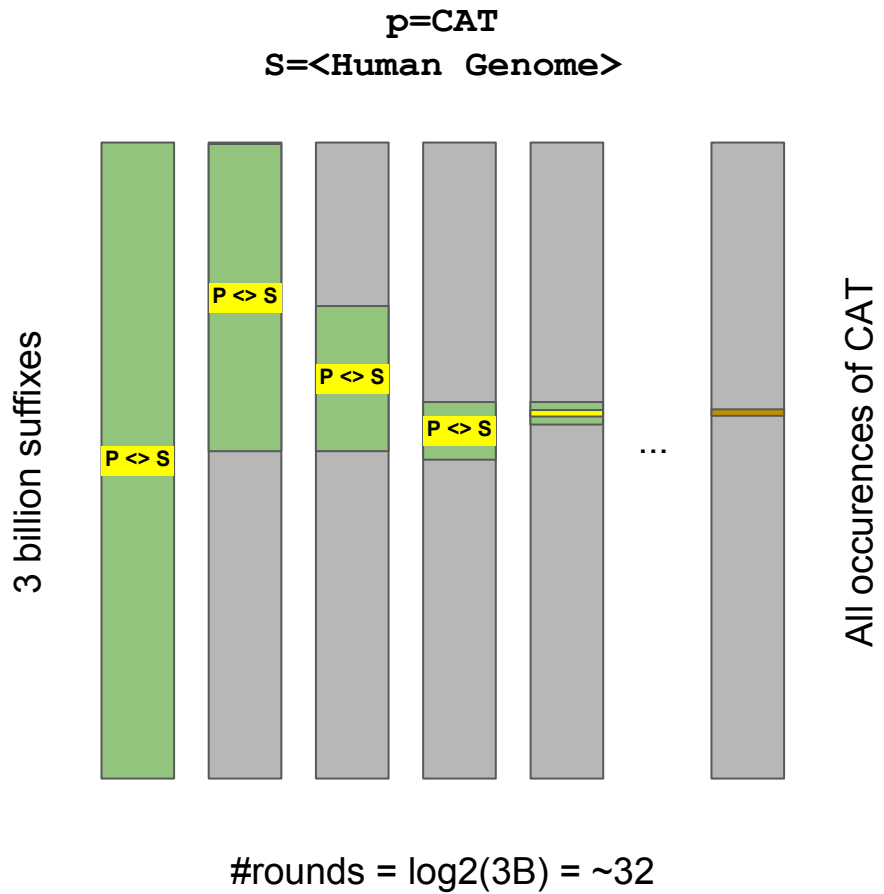
```
...
[15] ATCAT
[01] ATGCGCATGCTAGCATCAT
[07] ATGCTAGCATCAT
[17] CAT
[14] CATCAT
[00] CATGCGCATGCTAGCATCAT
[06] CATGCTAGCATCAT
[04] CGCATGCTAGCATCAT
[10] CTAGCATCAT
[13] GCATCAT
...
```

# Searching the Suffix Array

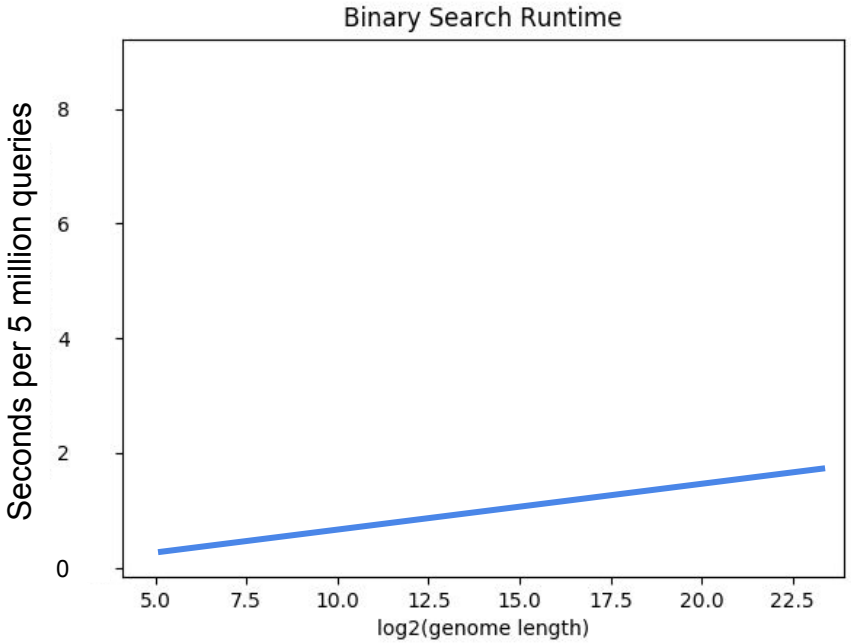
***Binary search-like algorithm:***

- Check middle of suffix array and compare suffix starting there to query - if too high check lower half of suffix array, and if too low check upper half
- Repeat this process to make the search range smaller until there is only one position left

Because search range is cut in half in each round, it can quickly identify the correct rows

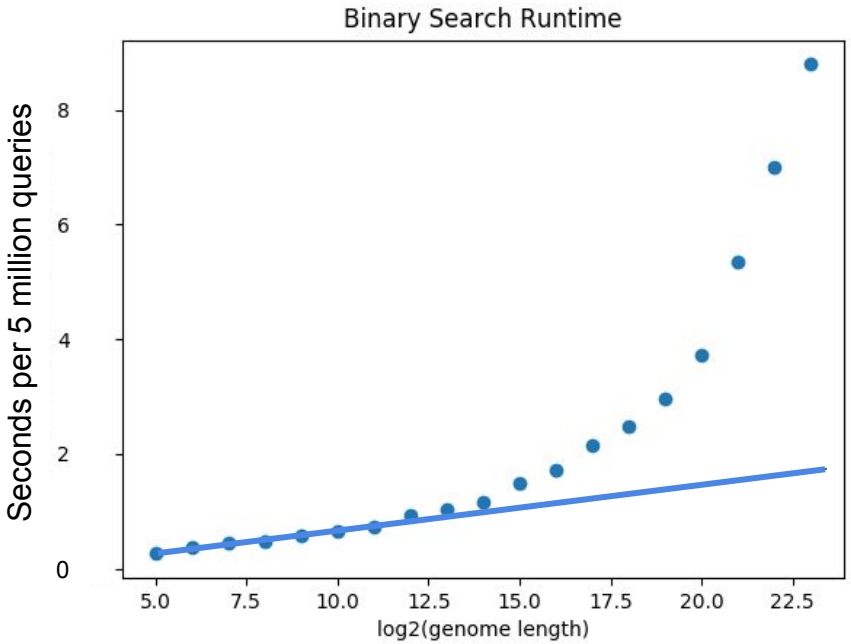


# Caching and Binary Search



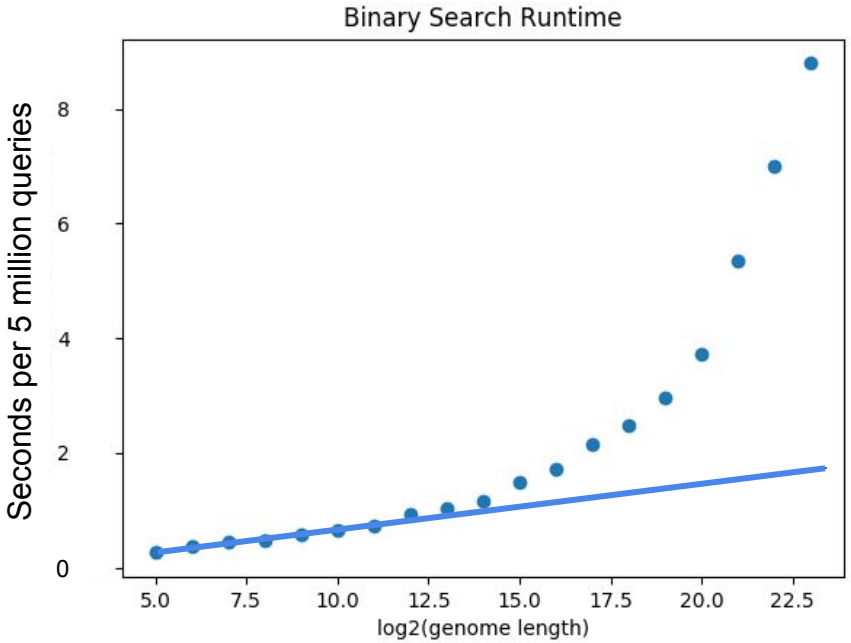

***In theory,*** searching should scale linearly with log2 of the genome size

# Caching and Binary Search



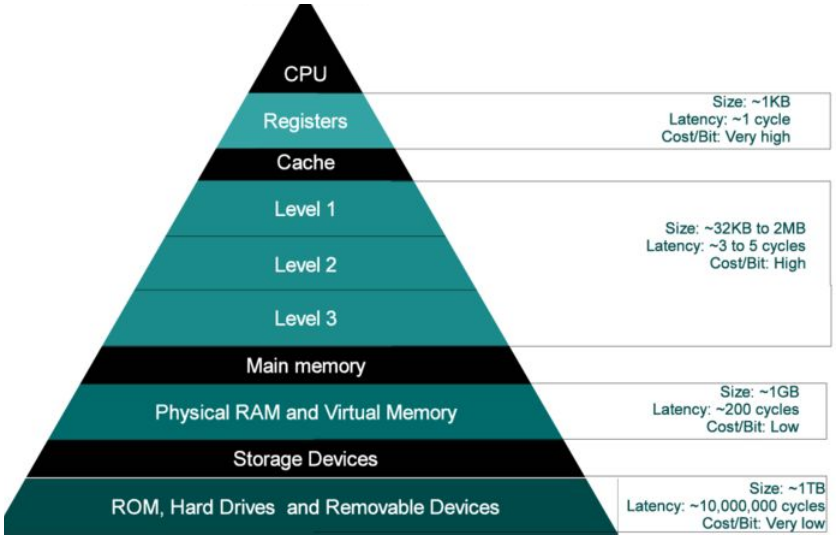

***In practice,*** searching is much slower for large genome sizes

# Caching and Binary Search



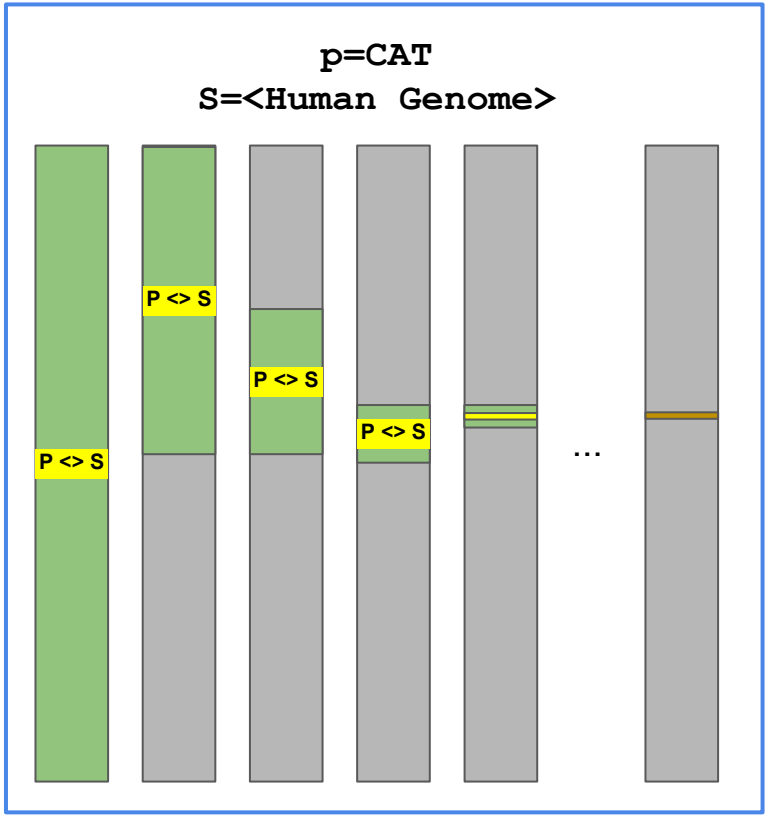

***In practice,*** searching is much slower for large genome sizes

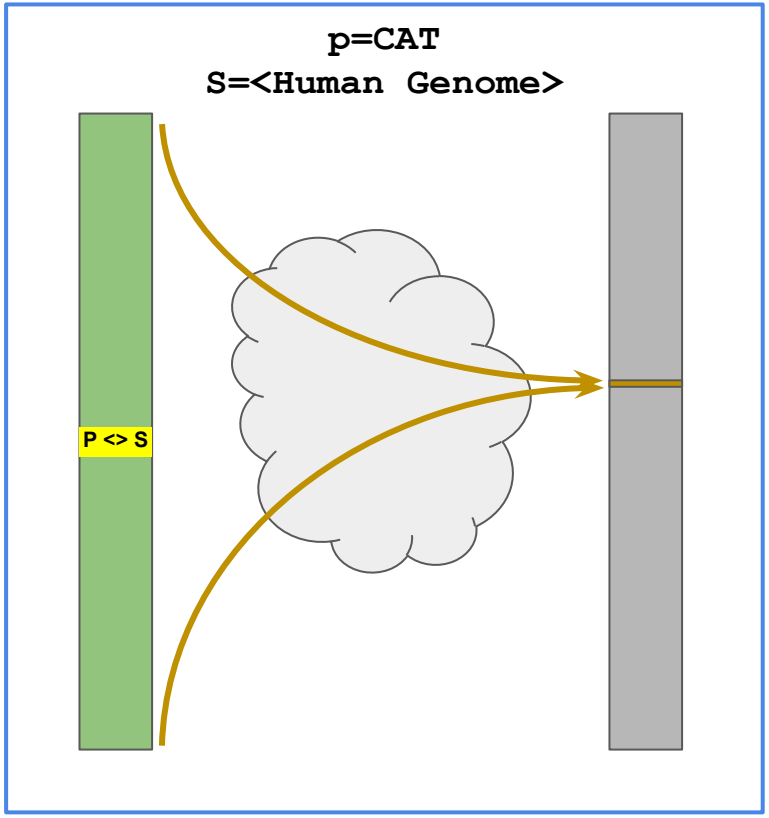
***Memory Hierarchy***




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

# Suffix Array Prediction



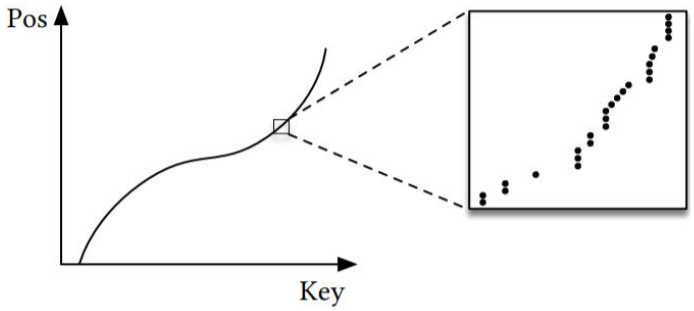

***What if instead of a slow algorithmic approach to find the correct rows, we could somehow quickly guess/predict the correct rows?***

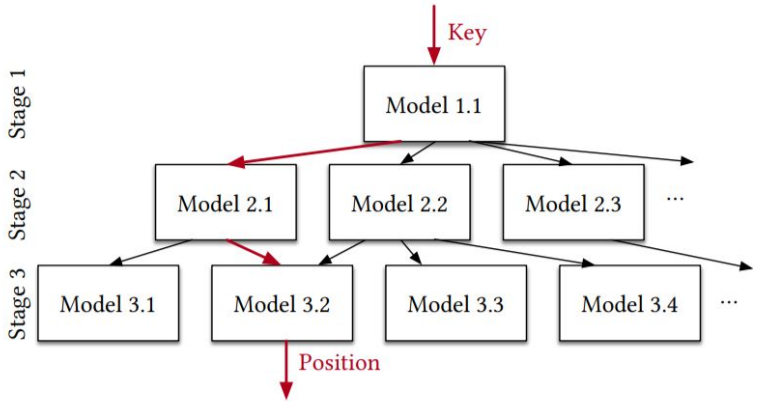
# Learned Index Structures

Researchers at Google using neural networks to replace 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

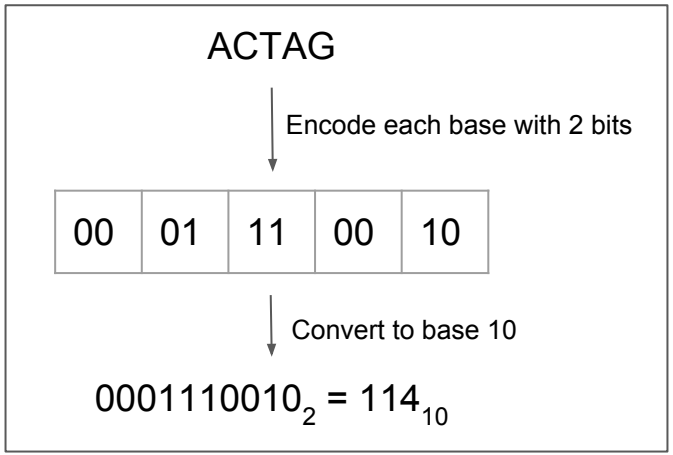
Compute the maximum error E = |predicted position - actual position| among all points in data structure. Then, narrow search to within E of predicted value.



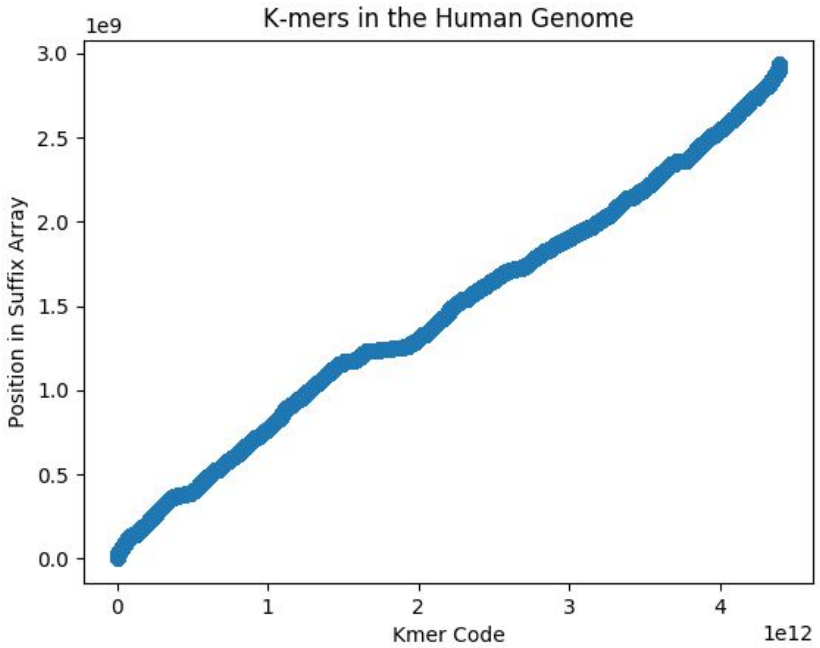




Kraska et. al. “The Case for Learned Index Structures”. SIGMOD 2018.

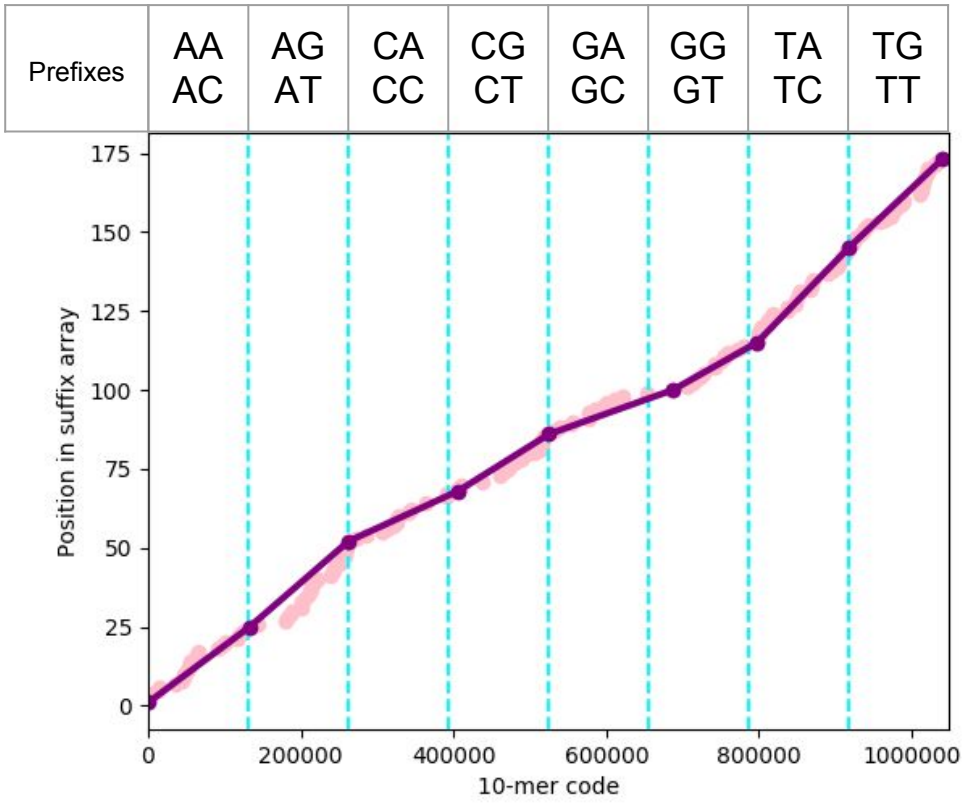
# Suffix Array Search as a Prediction Task



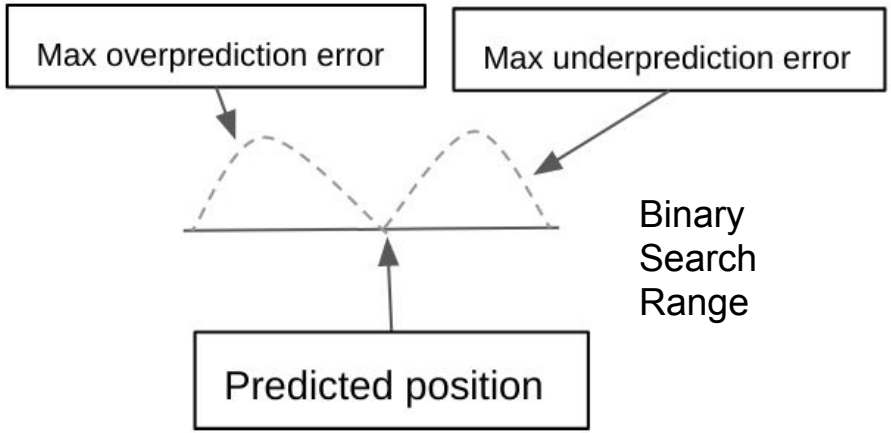

**Goal**: Given a query string and a suffix array, predict the suffix array position where the suffix begins with that query string



# Piecewise Linear Prediction Scheme



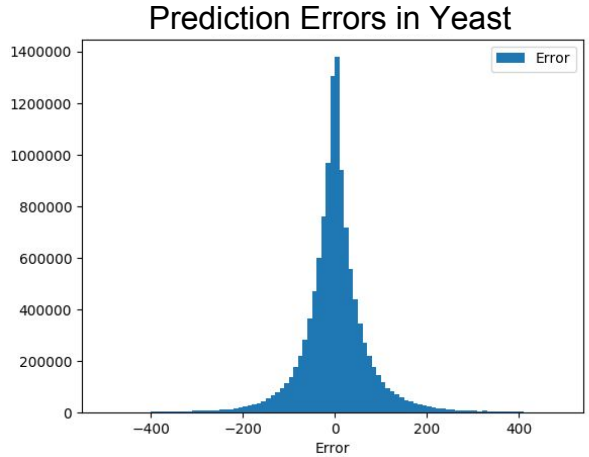


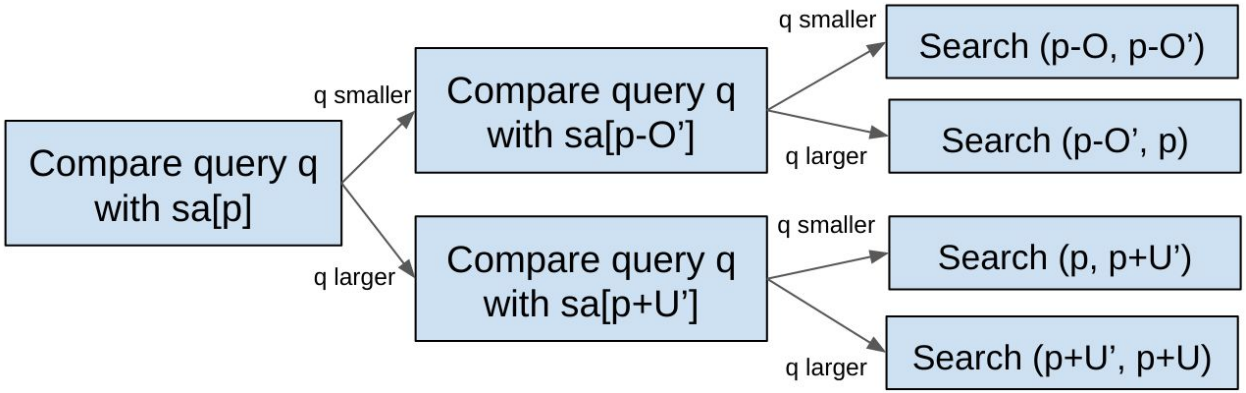


# Optimizing the Common Case

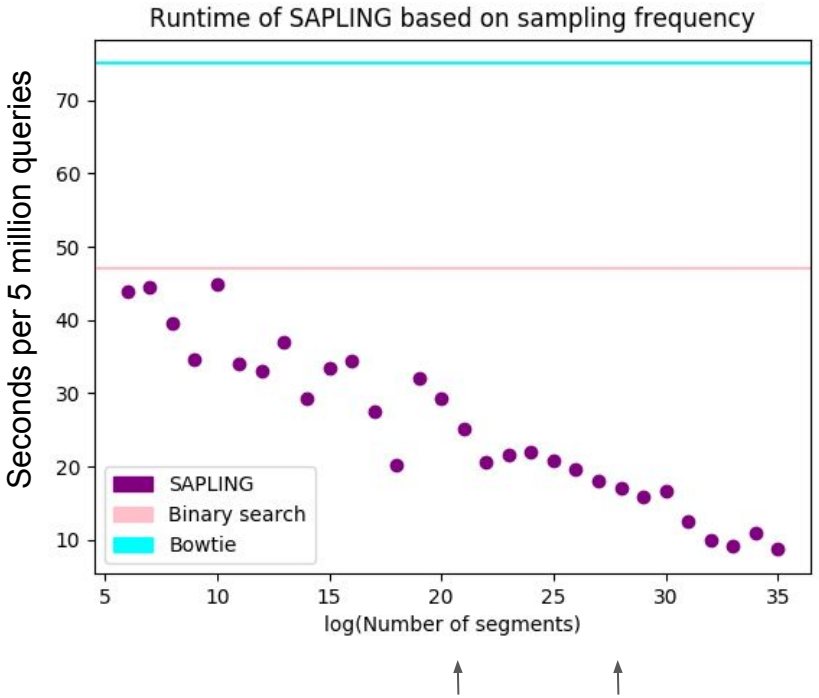
Keep track of not only maximum error but also 95th percentile of error distribution

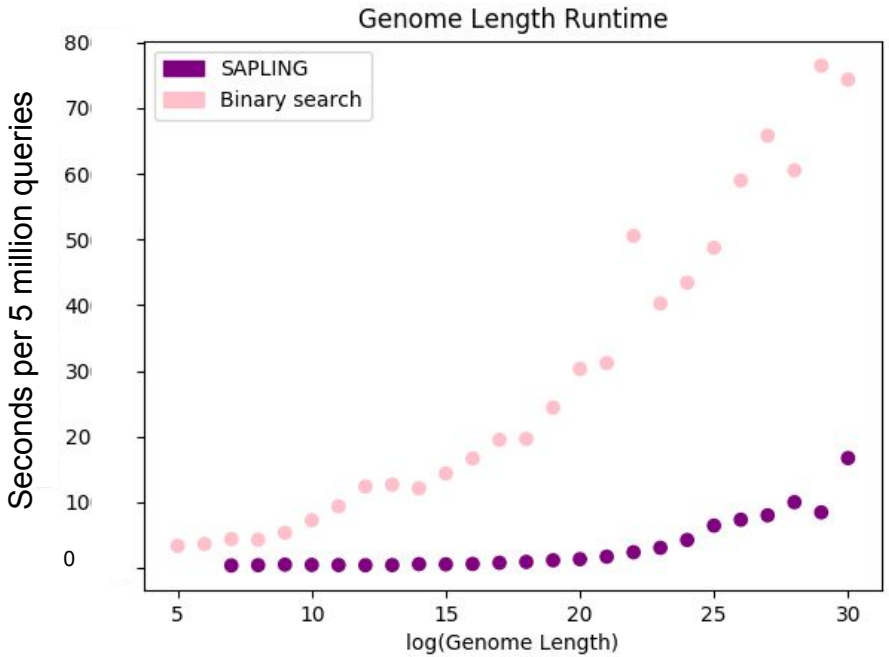
Reduces search space by about an order of magnitude in 95% of cases



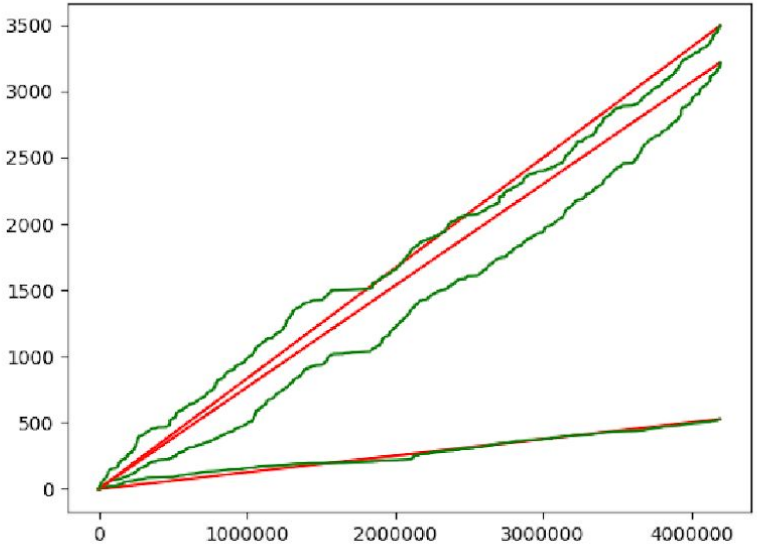
# Performance Results



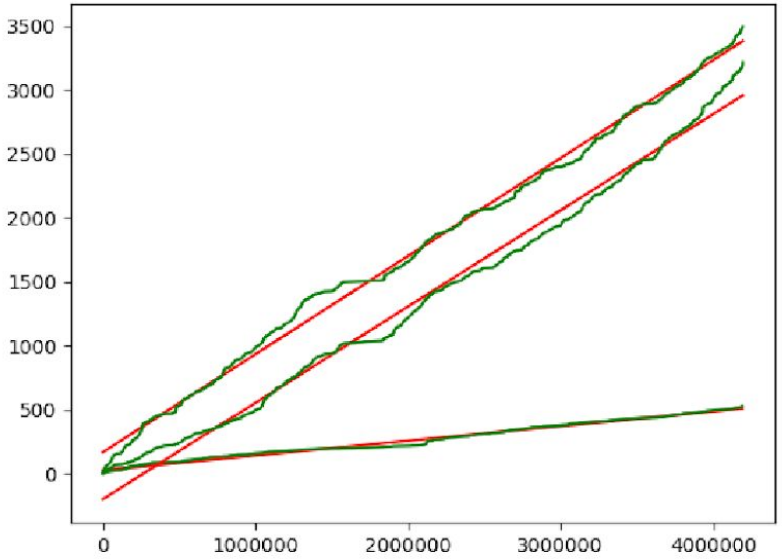



# Ongoing Work - Better Functions

**Piecewise linear: average error = 276.188**



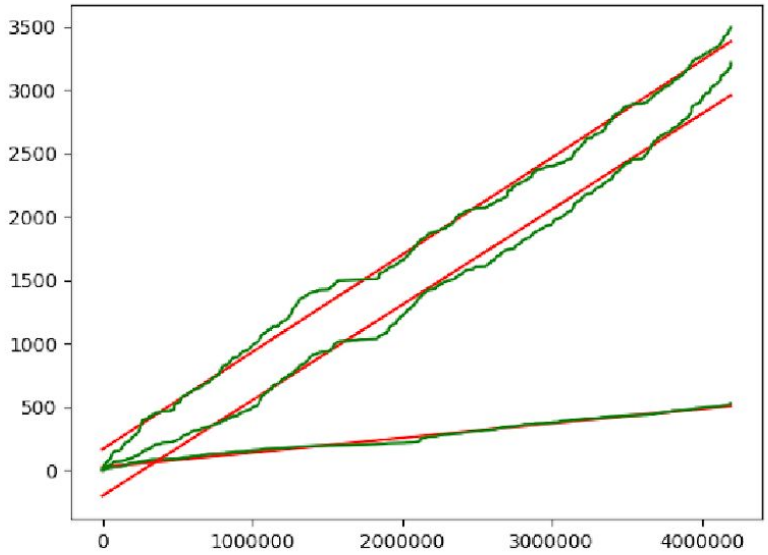

**Linear regression: average error = 158.831**



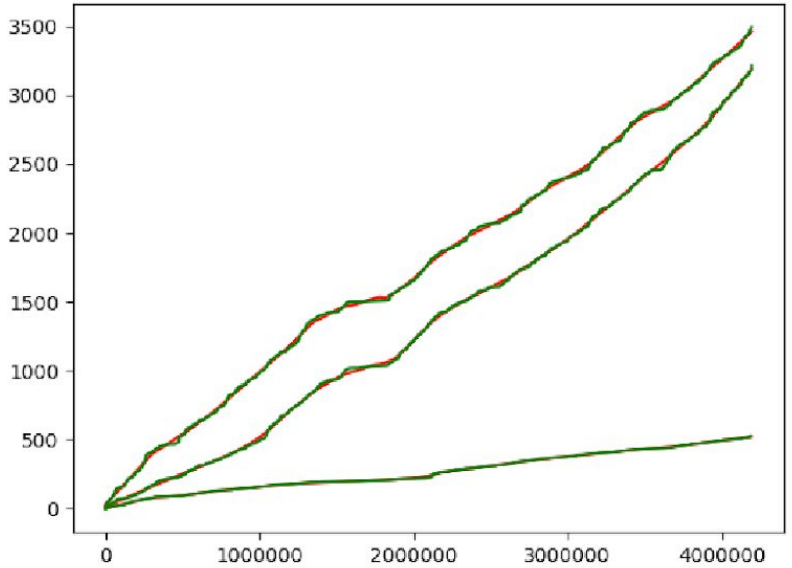

Suffix Array Position
Predicted Position

# Ongoing Work - Better Functions

**Quadratic: average error = 149.185**




**Lowess: average error = 13.747**




Suffix Array Position
Predicted Position

# Conclusion

***SAPLING allows faster string searching than existing approaches***

- Scales well with genome length and could be used for searching large collections of genomes, e.g. metagenomics search
- Technique of treating data structure lookups as predictive function evaluation has the potential to speed up many other genomic data structures

***Future Work***

- Optimize SA Prediction Function
- Support in-exact and exact string matching
- Allow variable length queries

***Software Release***

- Open Source: https://github.com/mkirsche/sapling