Exact Solution for the Approximate Suffix-Prefix Overlap Problem

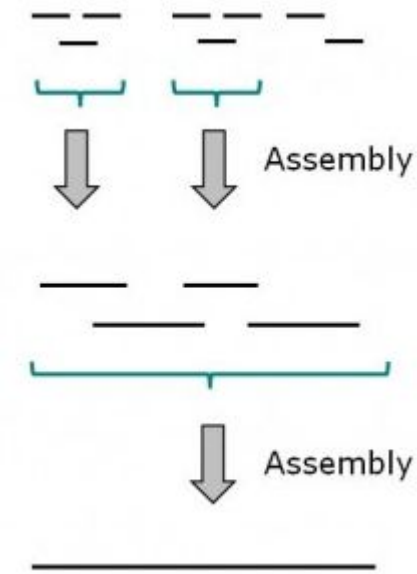
Christopher Esterhuyse
Thesis for B Computer Science
VU Amsterdam

Contents

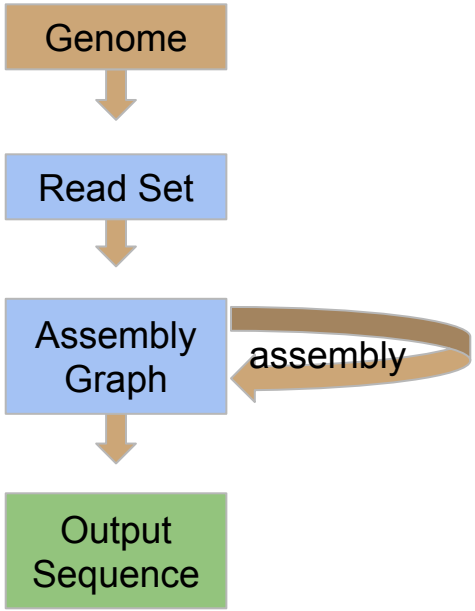
1. Context
   a. Sequence Assembly
   b. Sequence Assembly Pipeline
   c. Approximate Suffix-Prefix Overlap Problem
2. Existing Works
3. My Task
4. Solving ASPOP
   a. Using a Text Index
      (Interlude) Filter algorithm defined
   b. Suffix Filters
5. Suffix Filter Algorithms
   a. Välimäki
   b. Kucherov
6. Progress
   a. Implementation
   b. Extensions
   c. Experiments
      i. Phase 1
      ii. Phase 2
      iii. Phase 3
7. Future Work
8. Question Time

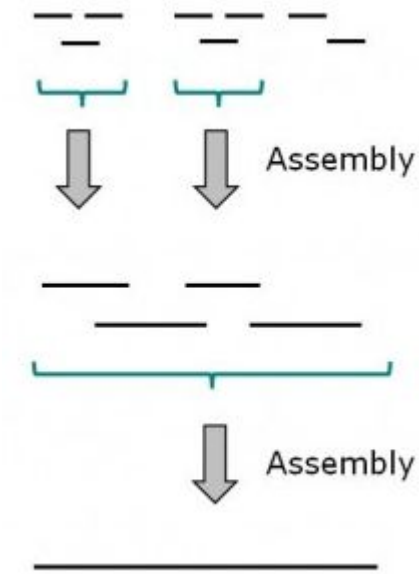
Context:
Sequence Assembly

- In-silico representation of a genome
- Can’t physically read the entire genome at once
- Need to assemble small reads into the entire genome

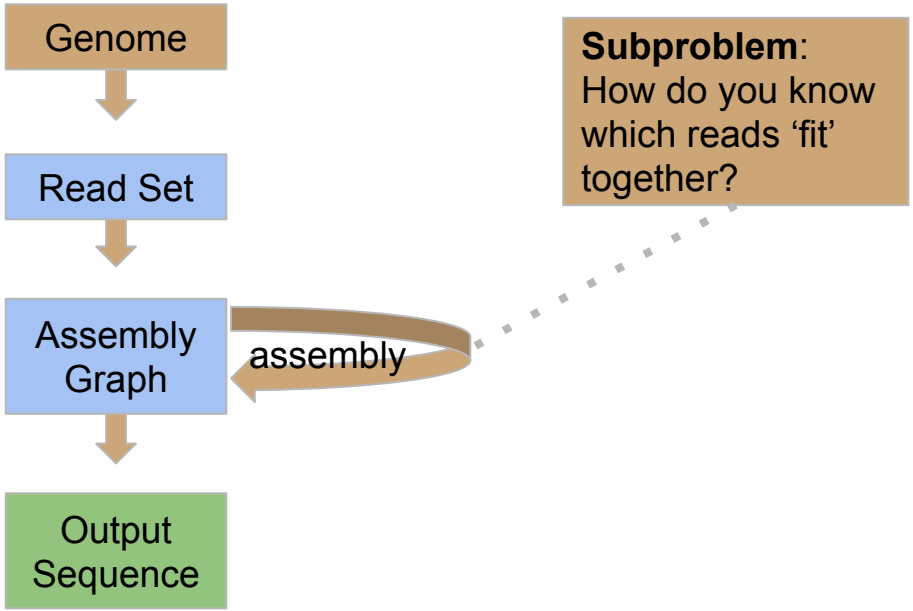


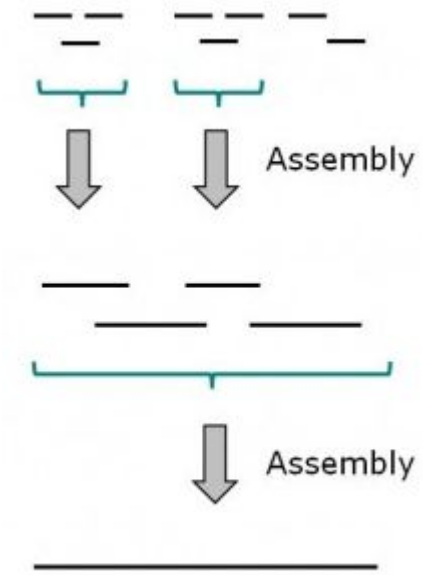
Context:
Sequence Assembly Pipeline





Context:
Sequence Assembly Pipeline



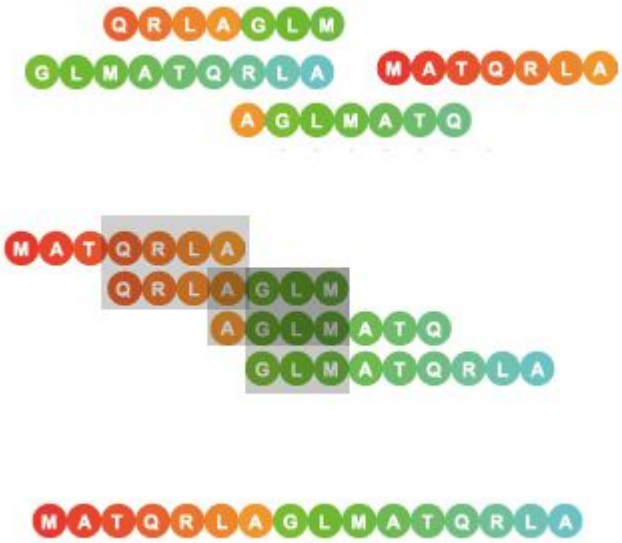


Context:
Approximate Suffix-Prefix Overlap Problem (ASPOP)

Subproblem of Sequence Assembly

- No knowledge of *where* a read is from
- Instead, guess which reads come from *common* locations (overlaps)

Given a set of **strings,**
return a set of **overlaps**

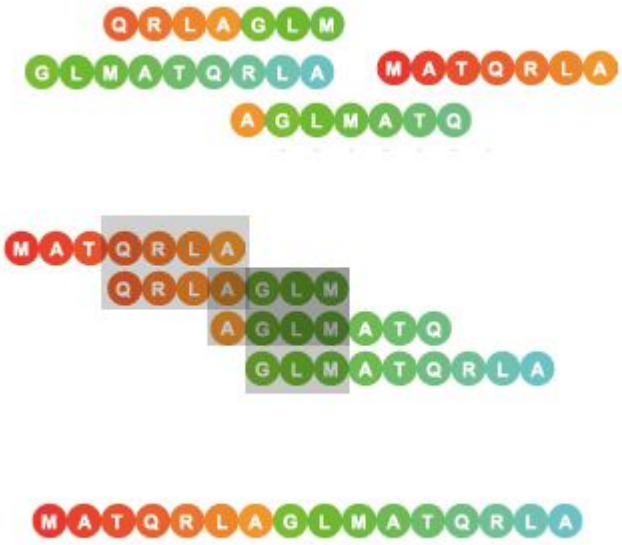


Context:
Approximate Suffix-Prefix Overlap Problem (ASPOP)

Read process is imperfect!

Overlaps must be allowed to be **approximate**
i.e. contain **errors**

TTGGGCAATGA
CCCCTTAGGC



Context:
Approximate Suffix-Prefix Overlap Problem (ASPOP)

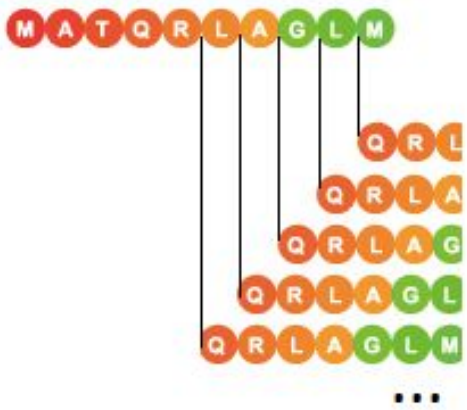
Heuristic solvers for the ASPOP exist

- Exact solution is prohibitively expensive

Can we solve it *exactly*?

Naive approach:

- For every pair of strings, check if any overlap of suffix-prefix is a *good solution*
  - *Moderate data set has 40 000 reads*
  - *This approach takes a long time!*

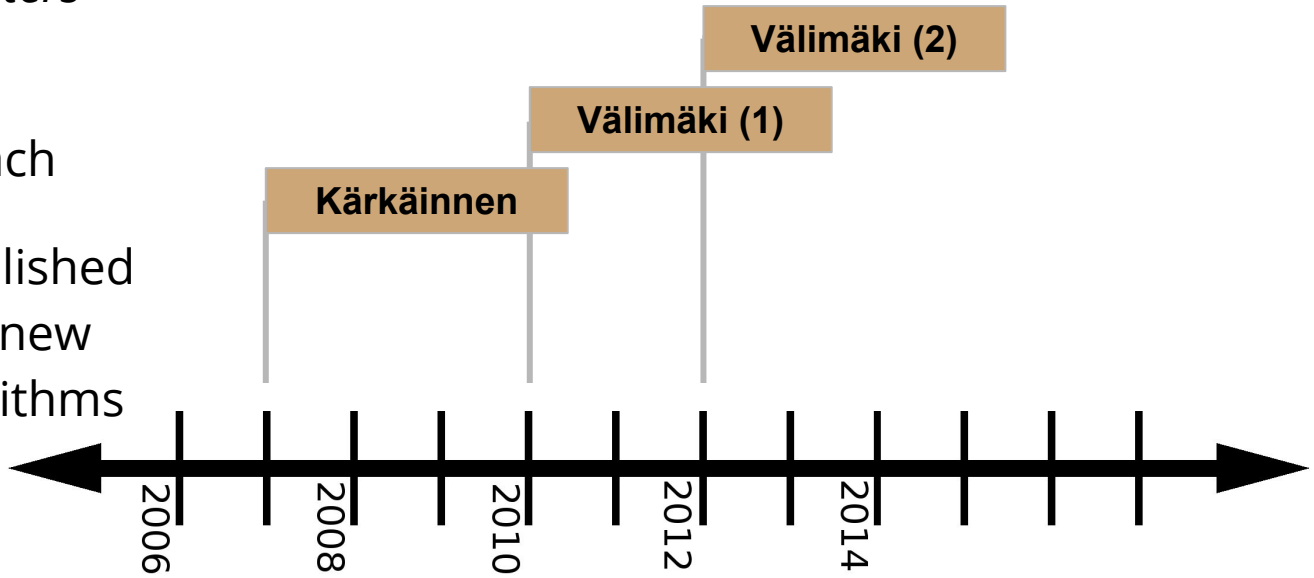


Existing Works

Many approaches are possible.
One approach: *suffix filters*

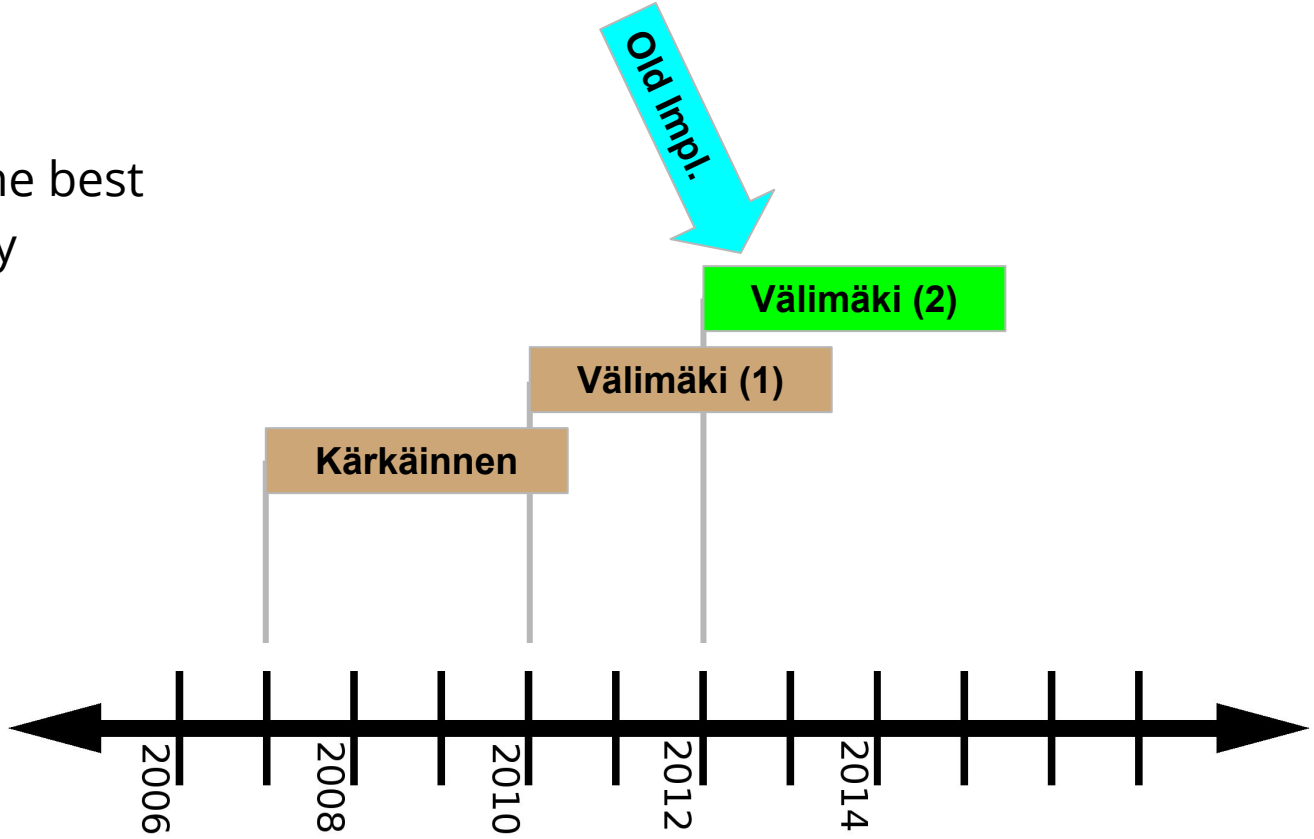
Kärkäinnen et al first introduced this approach

More papers were published since then, suggesting new faster suffix filter algorithms



Existing Works

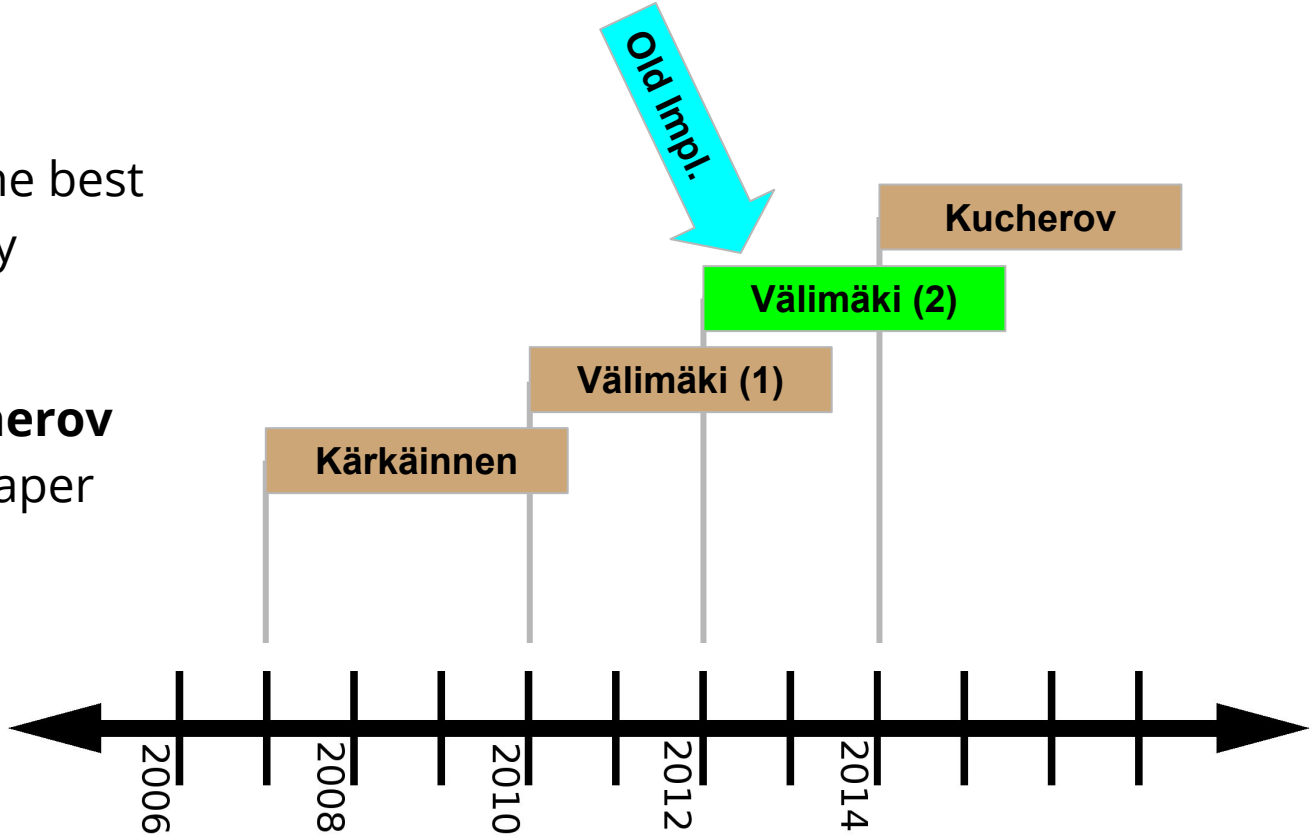
In-house solver uses the best algorithm as of 2012 by **Välimäki** et al.



Existing Works

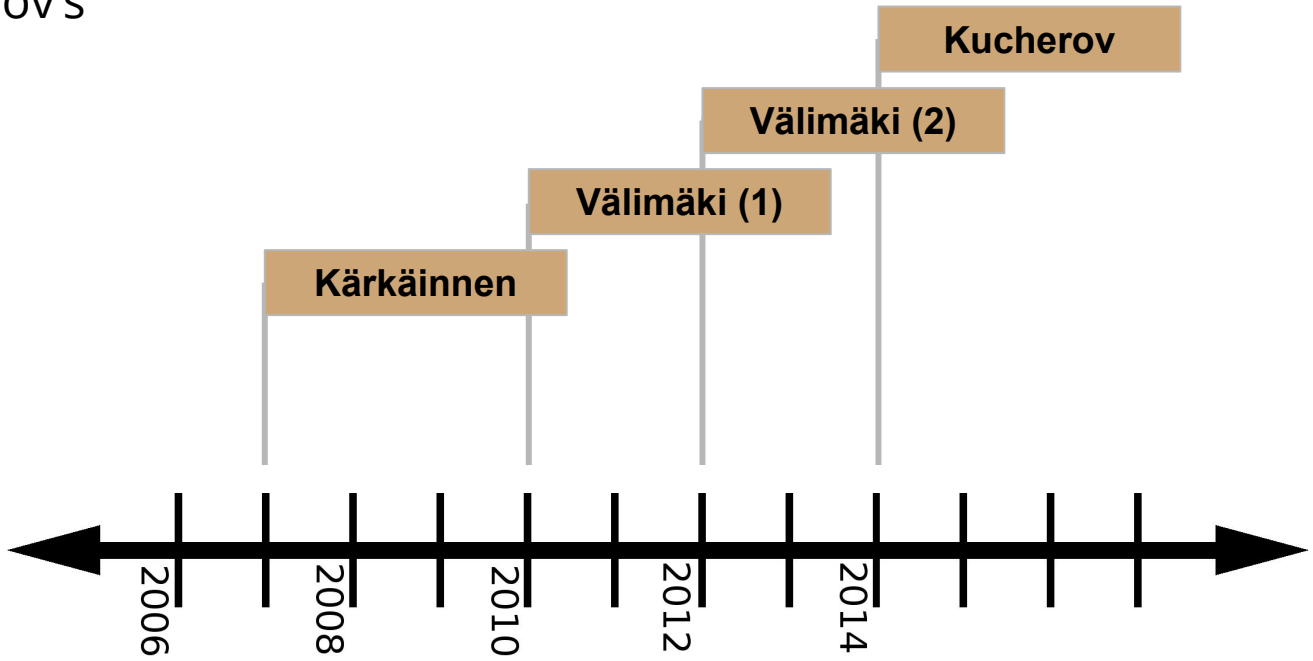
In-house solver uses the best algorithm as of 2012 by **Välimäki** et al.

In the meantime, **Kucherov** has published a new paper with a new algorithm.



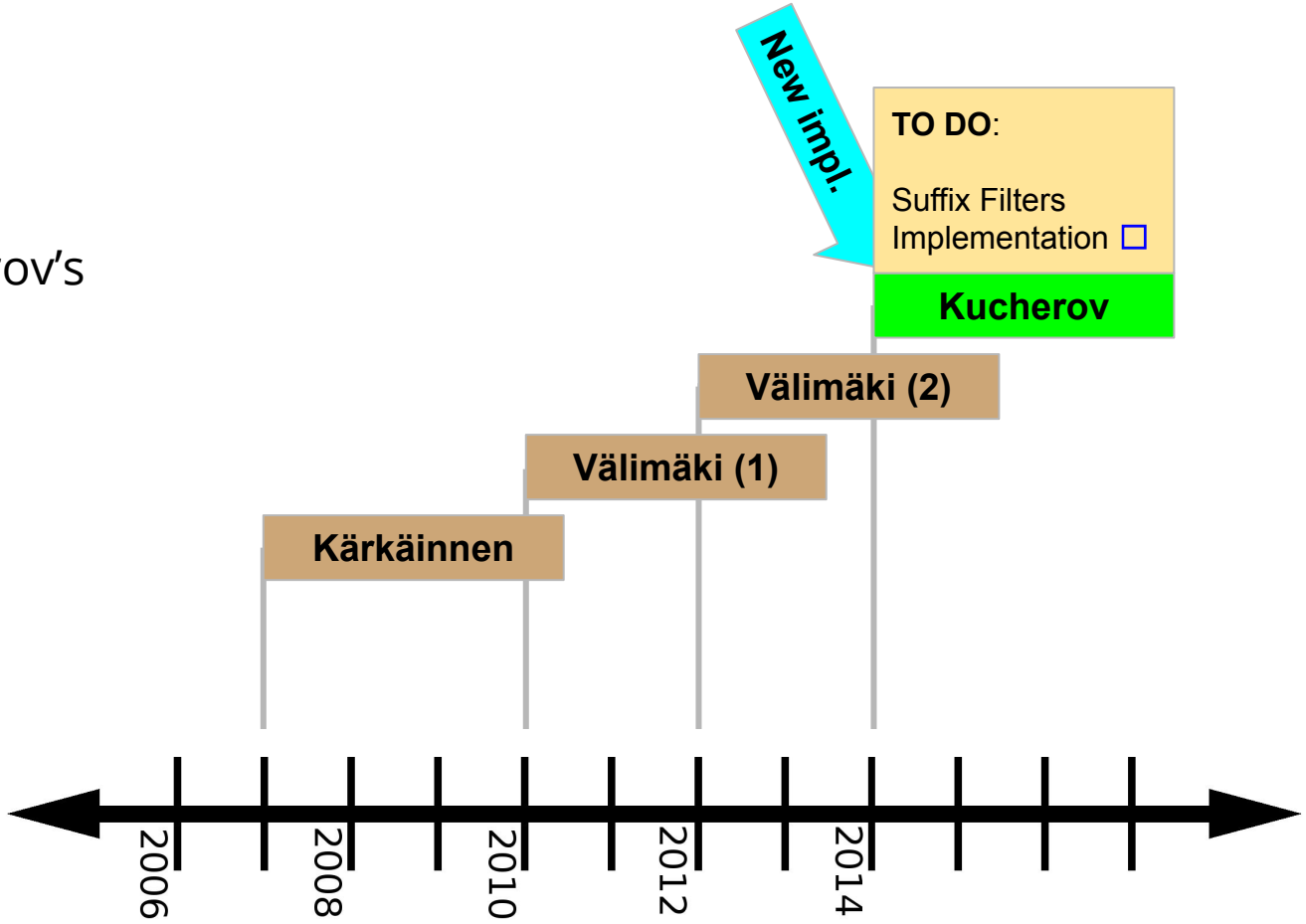
My Task

1. Implement Kucherov's algorithm



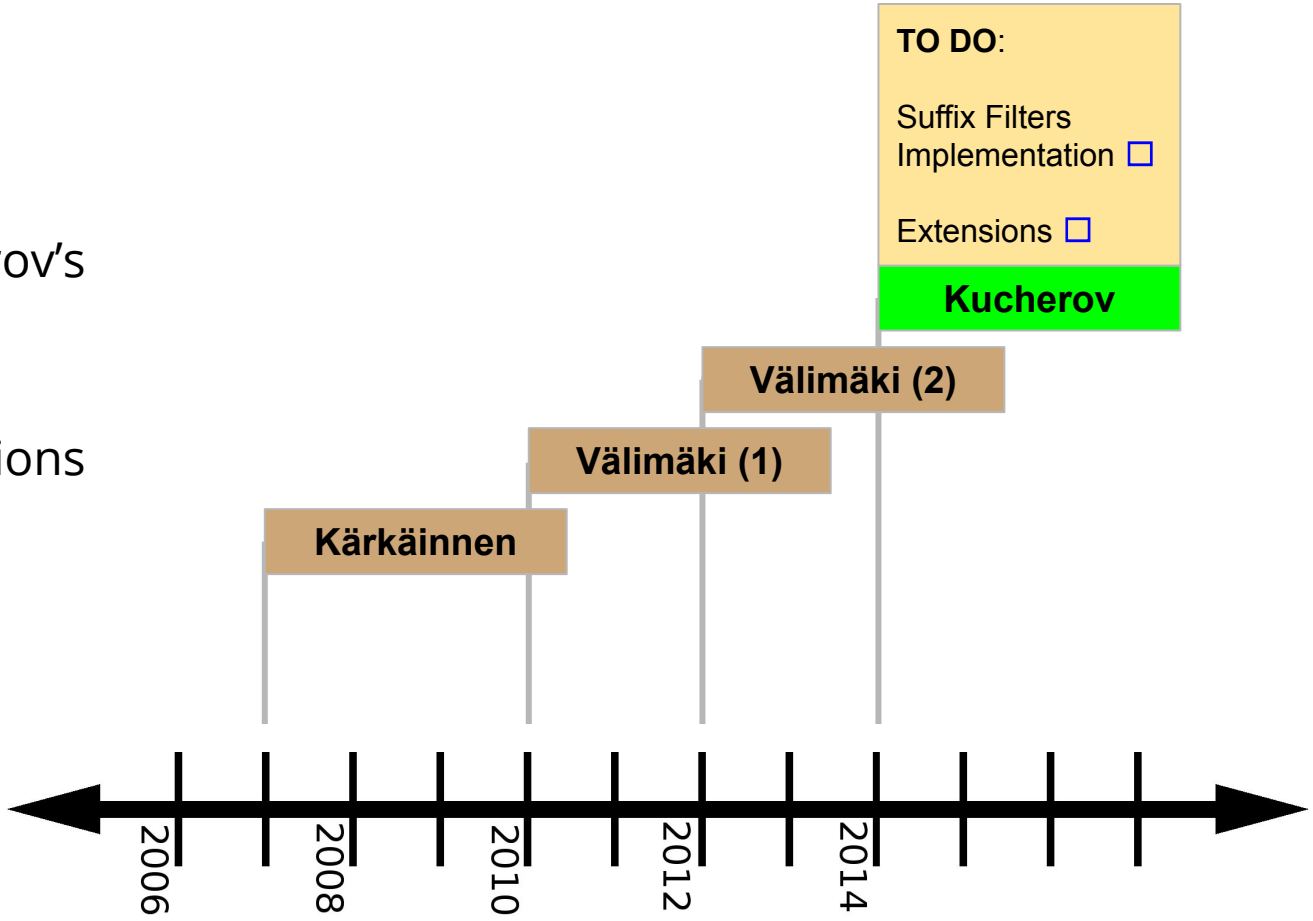
My Task

1. Implement Kucherov's algorithm



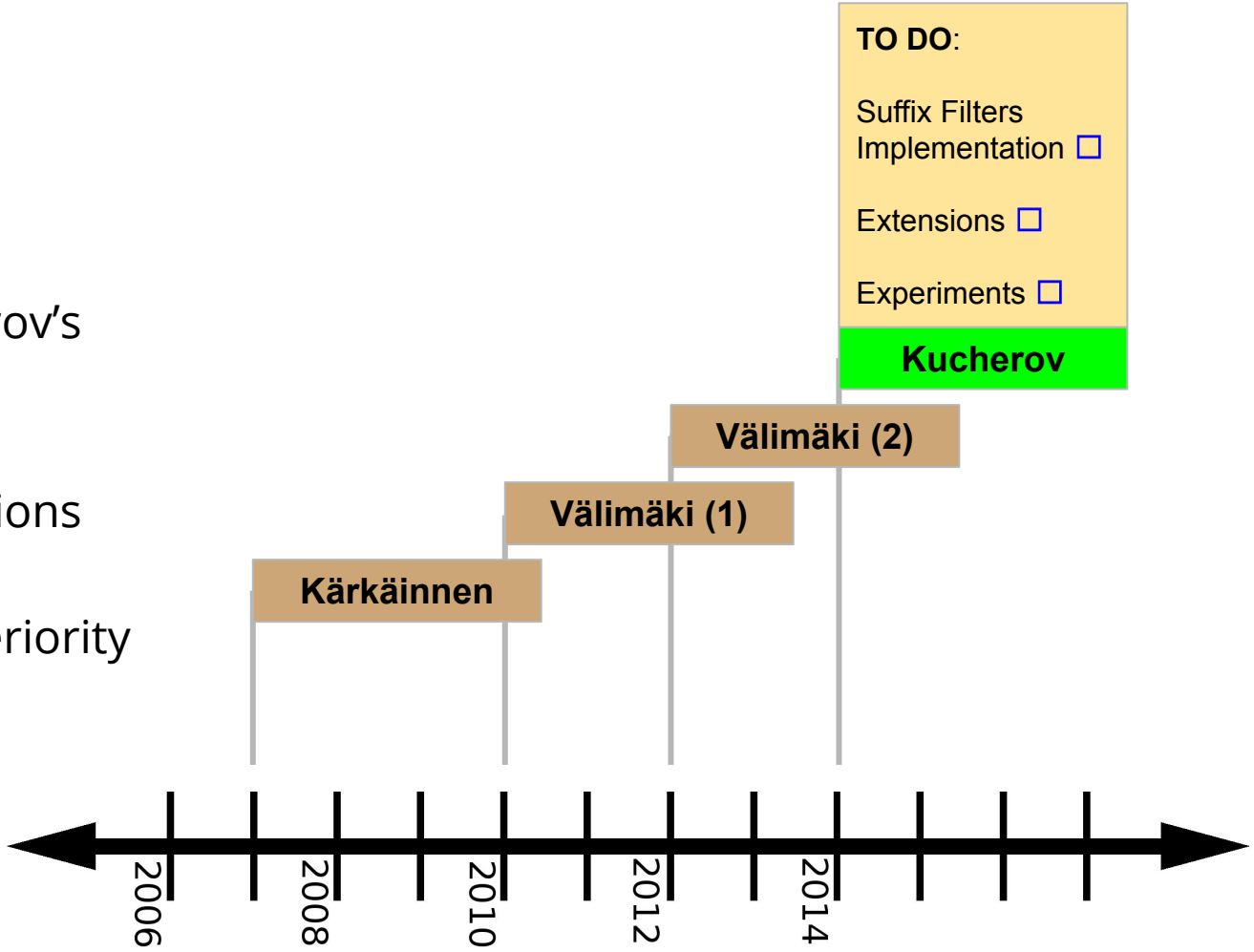
My Task

1. Implement Kucherov's algorithm
2. Implement extensions



My Task

1. Implement Kucherov's algorithm
2. Implement extensions
3. Demonstrate superiority experimentally



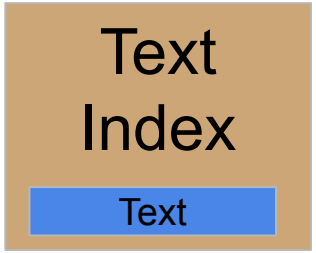
Solving ASPOP

- Naive solution is to check error of all overlaps pairwise
  - Too slow!

Solving ASPOP:
Using a text index

- Naive solution is to check error of all overlaps pairwise
  - Too slow!

- Take advantage of a **text index**. Find all matches of one string in <u>all others</u> at once.

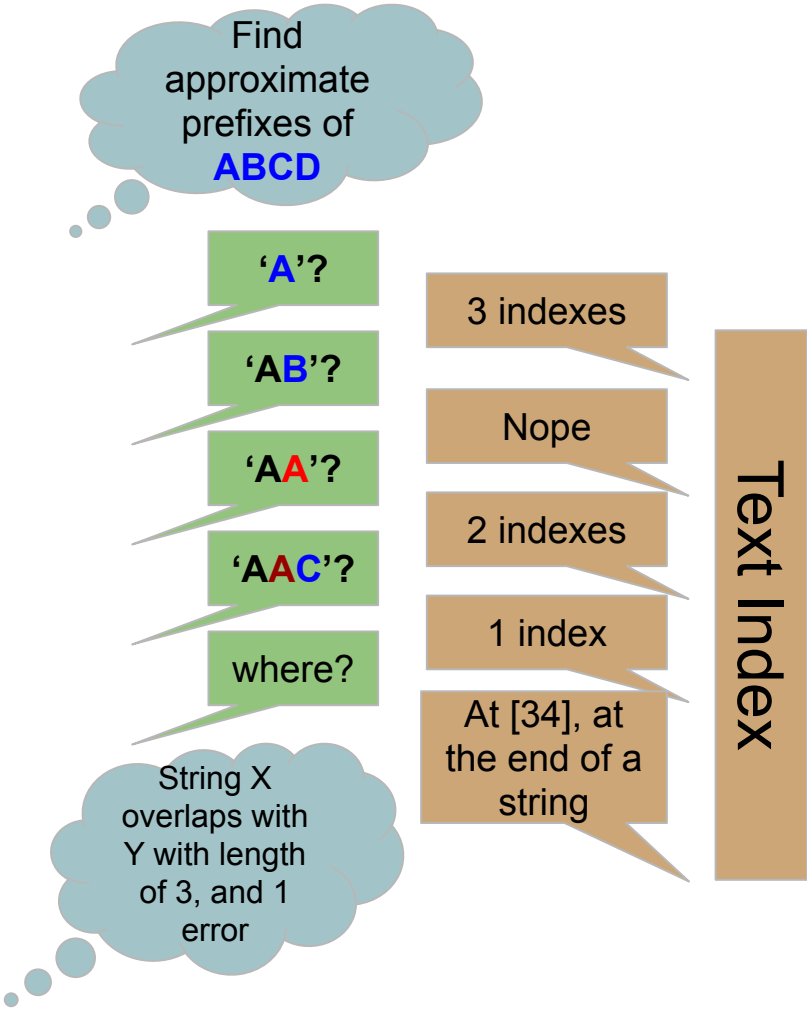


Question:
How to get **approximate** overlaps from **exact** queries?

Solving ASPOP: Using a text index

Approximate overlaps from **exact** queries:

- Index can only locate exact string/symbol occurrences.
- Get around this by strategically "asking" the index the right questions

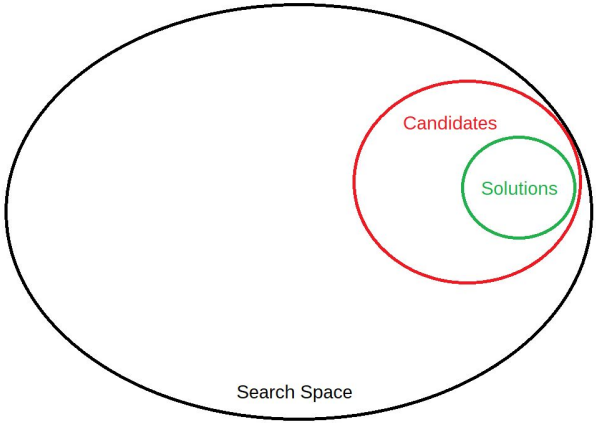


(Interlude):
Filter algorithm defined

Filter Algorithm:

- Break problem into 2 parts
  - Candidate generation step (aka search step)
  - Candidate verification step

- The sum of the two parts is faster than the naive approach



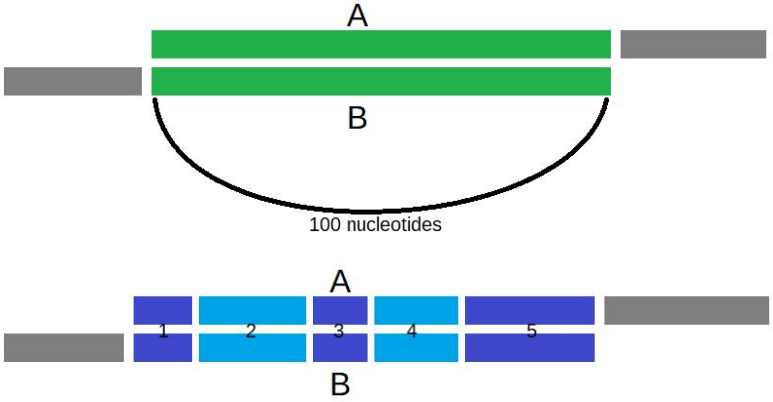
Solving ASPOP: Suffix filters

Observation:

Consider some overlap of length **100** between some A, B that we want to find.

- Say we have defined a maximum error rate of 4.0%

- Divide overlapping part into 5 blocks
  - Overlap ϵ solutions → at least 1 block has no errors.



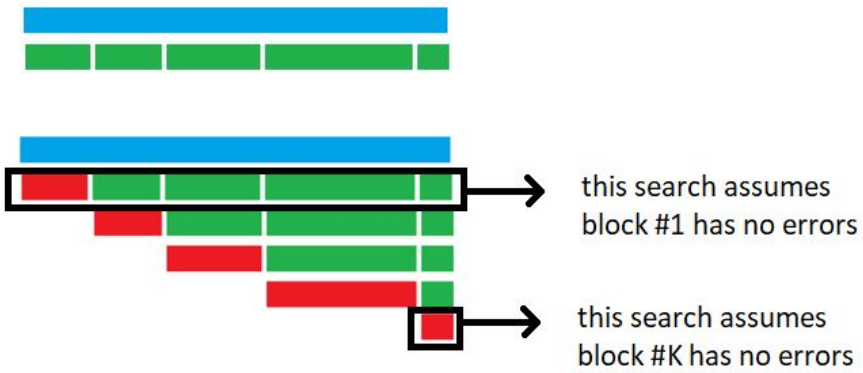
Solving ASPOP: Suffix filters

For each string **P** in the input read set:

1. Partition **P** into N blocks

2. Generate a set of assumptions
   - Block $\mathbf{P_1}$ has no errors
   - Block $\mathbf{P_2}$ has no errors

   …
3. Generate N searches of the index, each using 1 assumption.
   How the query is performed is dictated by a *filter* (~branch rules)



- If you partition **P** strategically, and your filters are lenient enough you are guaranteed that 1+ assumptions hold for every true solution → it will be found

- Total search cost is less than naive approach

Suffix Filter Algorithms: Välimäki

In their **1st** paper [V1]:

- Applied Kärkkäinen's suffix filters to the ASPOP

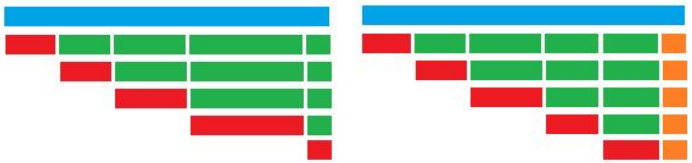
In their **2nd** paper [V2]:

- Identified a bottleneck in [V1] revolving around the **short last filters**
- Defined better filters which
  - Covered the same solutions
  - Avoided short last filter problem (faster)

Suffix Filter Algorithms: Kucherov

- Created a new *filtering scheme* that avoided the **short last filter** problem more efficiently than [V2]
  - by guaranteeing a redundant block so the short last filter could be avoided

- Defined a less wasteful *partitioning scheme*
  - which postpones search branching until the last possible moment



Progress:

TO DO:

Suffix Filters Implementation ☑

Extensions ☑

Experiments ☐

Progress:
Suffix Filters Implementation ✅

- New system is implemented in Rust
- Has **modes** for using available suffix filter algorithms
  - Including Kucherov's

Windows PowerShell

```
PS C:\Users\Christopher\Documents\GitHub\rust_overlaps> .\rust_overlaps.exe .
/data/5VM_real.p1.8000.fasta out.txt 0.02 20 -m=valimaki2
OK working.
OK wrote 349637 solutions.
OK completed in ~147 sec.
WORK NANOS:
        total    1174644525814
        search   723615174084     61.60%
        verif    451029351730     38.40%
PS C:\Users\Christopher\Documents\GitHub\rust_overlaps>
```

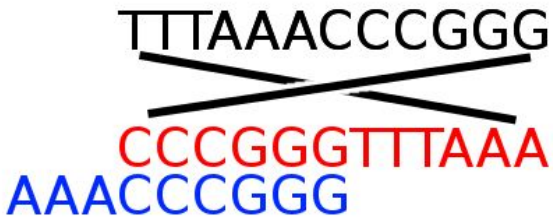
Windows PowerShell

```
PS C:\Users\Christopher\Documents\GitHub\rust_overlaps> .\rust_overlaps.exe .
/data/5VM_real.p1.8000.fasta out.txt 0.02 20 -m=kucherov_2
OK working.
OK wrote 349637 solutions.
OK completed in ~7 sec.
WORK NANOS:
        total    56851722587
        search   41494460592      72.99%
        verif    15357261995      27.01%
PS C:\Users\Christopher\Documents\GitHub\rust_overlaps>
```

Progress:
Algorithm extensions ✅

- Program supports all three extensions
- Each extension (when enabled) will find new types of solutions
  - **Reversals** : inverted overlaps
  - **Inclusions** : strings within each other
  - **Indels** : edit distance measure of error
- Each requires the algorithm to be adapted and elaborated

Reversals



Inclusions

Indels

Progress:
Experiments Phase 1 ✅

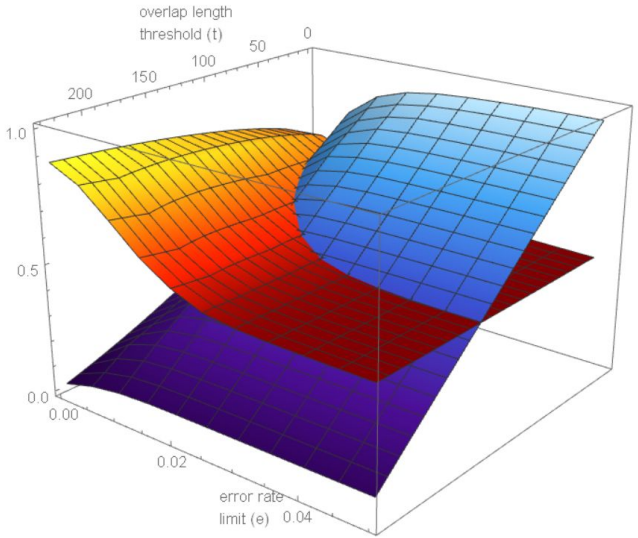
Seeks to identify relationship

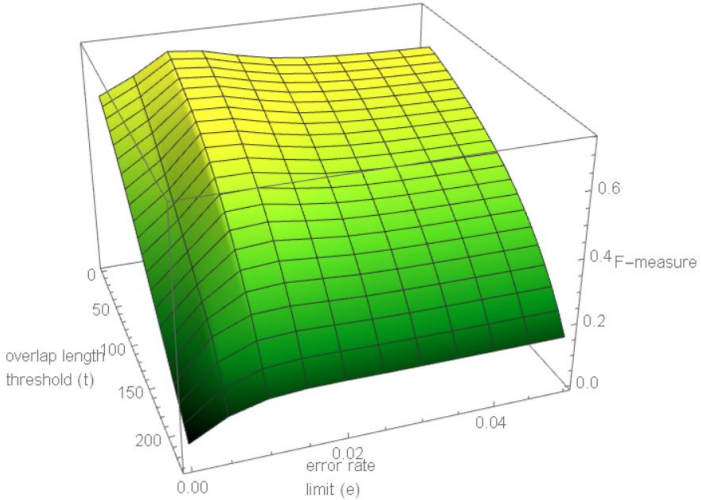
Input parameters

→ output quality (F-measure)

Observations:

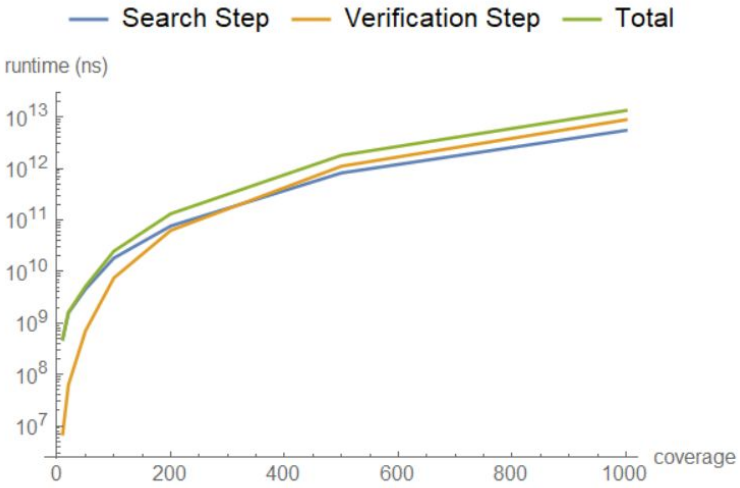
- Lower overlap-length threshold is consistently beneficial
- Decreasing precision plateaus towards higher error rate limits
- Settled on e=1.2% t=80



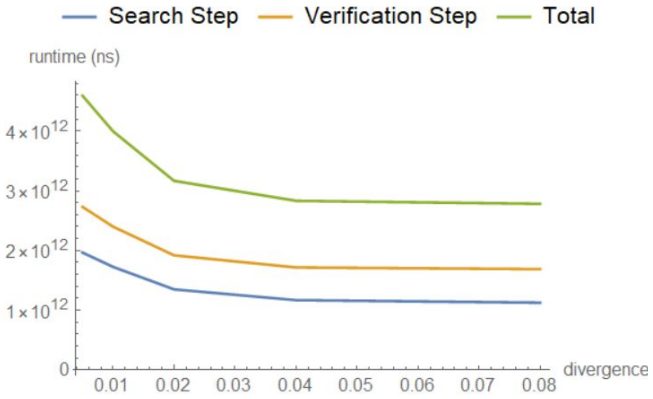


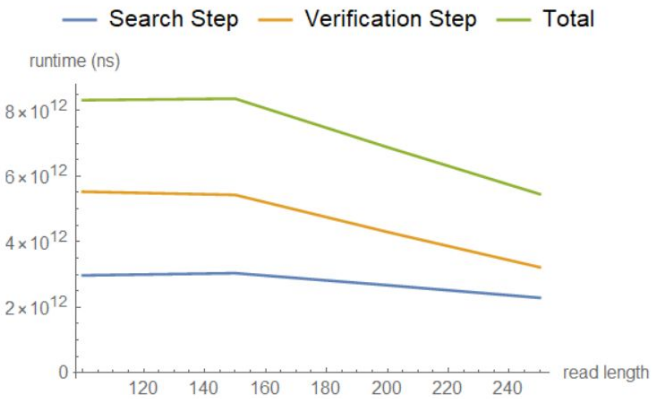
Progress:
Experiments Phase 2 □

Seeks to identify
Data set properties
→ runtime



Partitioned per **step**



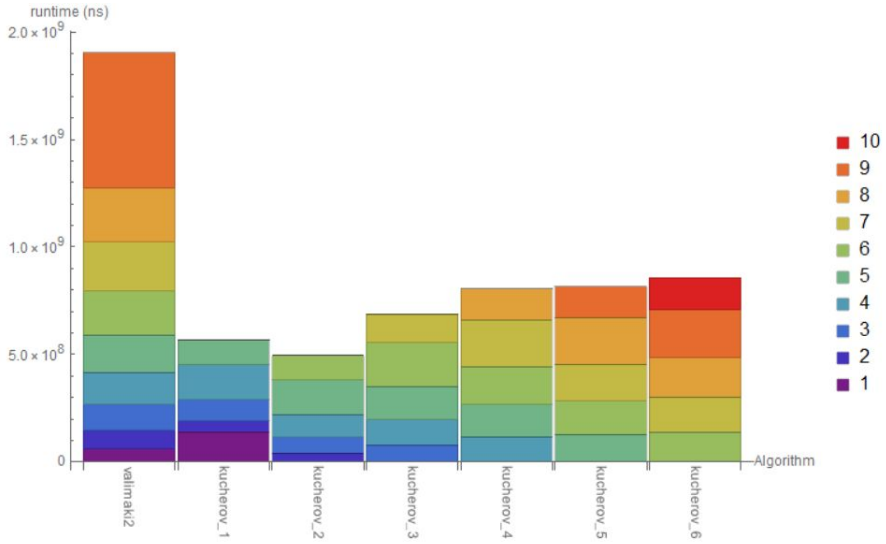


Progress:
Experiments Phase 2 □

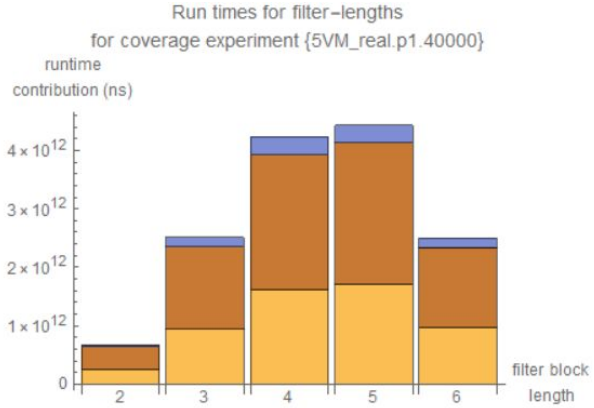
Seeks to identify

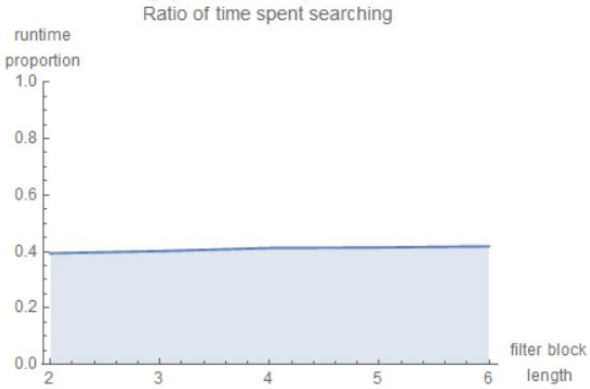
Data set properties

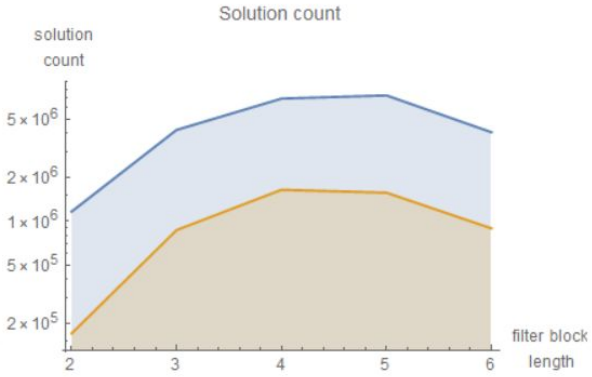
→ runtime



Partitioned per **filter**







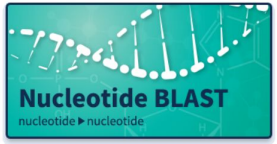
Progress:
Experiments Phase 3 □

Compare system performance in

- Output quality
- Runtime

As compared to existing solvers such as

- BLAST

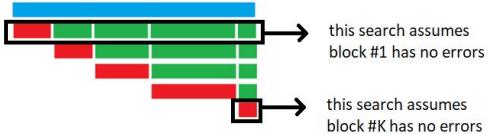


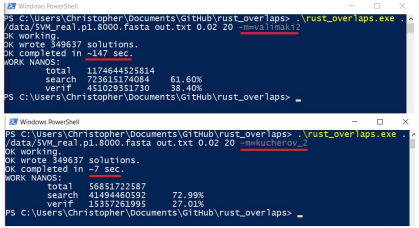
Future Work

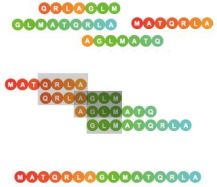
- Explore necessity of indels
  - Upside of using them: More solutions per coverage
  - Downside of using them: Slower runtime
- Optimize branching process for indels
  - Increase speed of indel-enabled solves

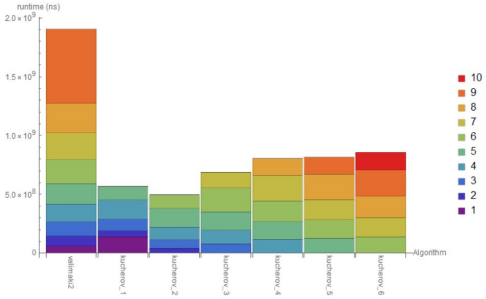
TTTAA • ACCCGGG
TGGGTTTAAAACCC

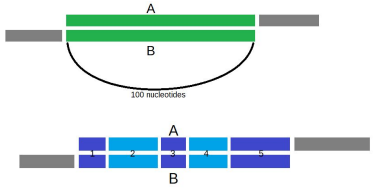
Question Time

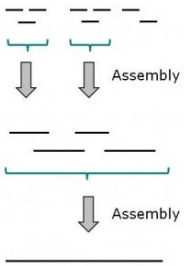


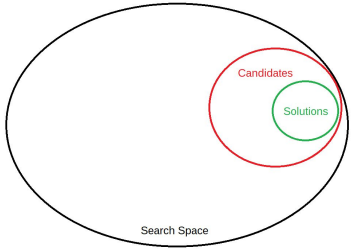


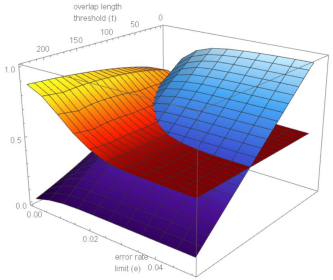


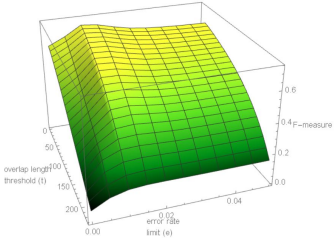


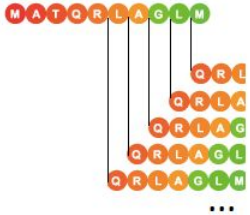


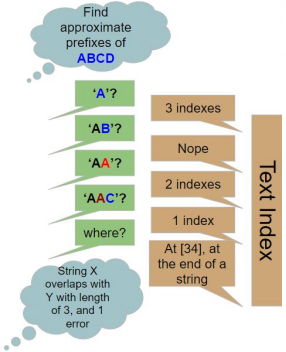


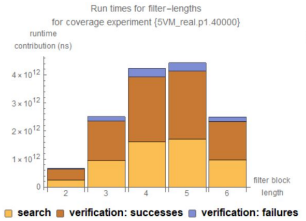




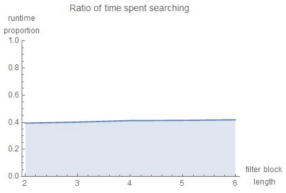




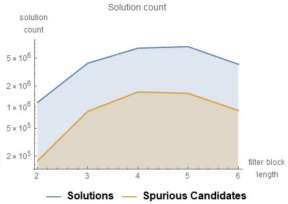




(a)



(b)



(c)

TTTAAACCCGGG
AATTTAAACCCGGGAA

TTTAA • ACCCGGG
TGGGTTTAAAACCC

TTTAAACCCGGG
CCCGGGTTTAAA
AAACCCGGG