Optimal Seed Solver: Optimizing Seed Selection in Read Mapping

Carnegie Mellon SAFARI — Bilkent University

Hongyi Xin[1], Richard Zhu[1], Sunny Nahar, John Emmons[1], Gennady Pekhimenko[1], Carl Kingsford[1], Can Alkan[2], Onur Mutlu[1]

[1] Departments of Computer Science and Electrical and Computer Engineering, Carnegie Mellon University, Pittsburgh, PA, USA

[2] Dept. of Computer Engineering, Bilkent University, Ankara, Turkey

Problem:

- NGS mappers can be divided into two categories: **backtrack based** vs. **seed-and-extend based**
  1. Backtrack based mappers (i.e. bwa, bowtie2) find **the best mappings fast** but **lose high-error mappings**
  2. Seed-and-extend based mappers (i.e., mrfast, shrimp, RazerS3) finds **all mappings** but **waste resources** on rejecting **incorrect mappings**
- **Problem:** seed-and-extend mappers select **high frequency seeds**
- **Our goal:** increase the efficiency of seed-and-extend based mappers by selecting the set of **least frequent** ***e+1*** seeds with **linear complexity**

1

The core dynamic-programming algorithm of OSS (OSS-DP)

3

- **Assumption:** the frequency of any single seed of the read is already known
- **Baseline:** enumerate all possible seed combinations, **$O(L^{e+1})$ possibilities**
- **OSS:** reduce the complexity to **O(e*L)**
- **Induction:** *m* seeds → *m+1* seeds
  1. Assuming the **least frequent** *m* seeds are already known for **any** substring of the read, ***R***
  2. For any substring, ***S***, it can then be divided into two parts by a divider, ***P***: an *m*-seed part and an *1*-seed part
  3. The least frequent *m+1* seeds of ***S*** can be found by moving the divider, ***P***, |***S***| times and select the **optimal divider** with the **minimum** total seed frequency
- **Insight:** consecutive optimal seeds of the read **must also be the optimal seeds of the substring** containing them (Fig 1)

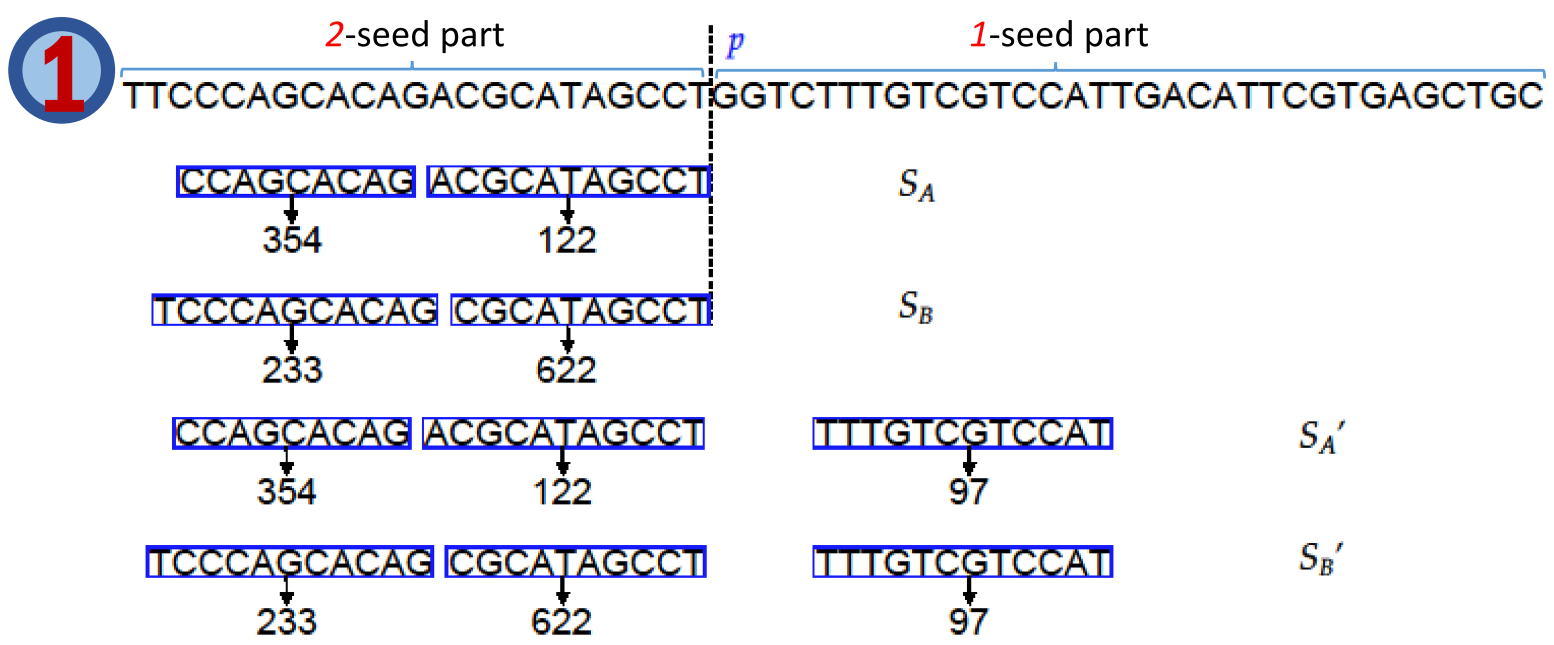

Fig 1: S_A and S_B are two combinations that occupies the same amount of letters. The total seed frequency of S_A is smaller. In this case, it is easy to prove that the total seed frequency of $S_{A'}$ will also be smaller than $S_{B'}$

Early Divider Termination (EDT)

5

- **ODC** confines the right bound of the optimal divider of a substring
- **Goal:** introduce a left bound
- **Key observation:** longer substrings have equal or less total seed frequency
- **Key idea:** move the divider, ***P***, from right to left, stop when the **frequency increase** of the left part **outweighs** the **total frequency** of the right part (Fig 3)
- **Key result:** with **ODC** and **EDT**, the **empirical** average number of comparisons to find the optimal divider of a substring is reduced to **5.25**

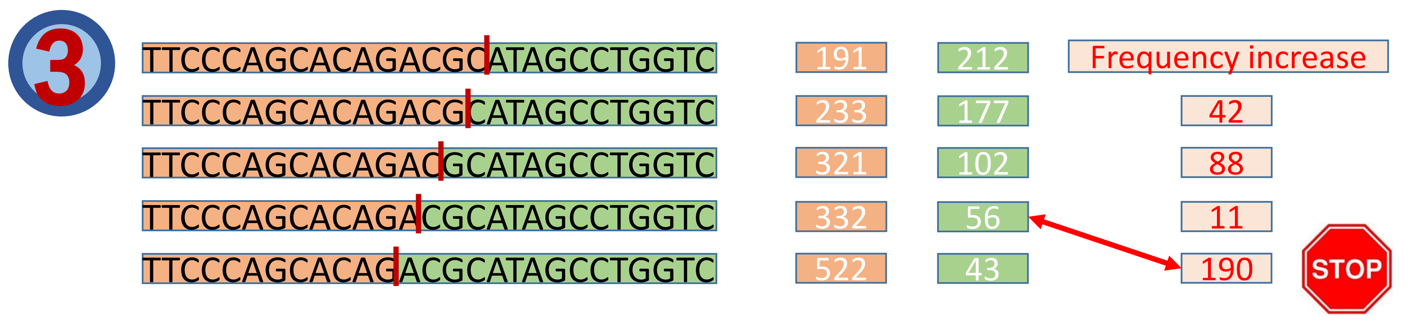

Fig 3: **EDT** in action. When the frequency increase of the left part outweighs the optimal *1*-seed frequency of the right part, STOP.

Conclusion and future work

7

- **Conclusion:**
  1. **OSS** finds the least frequent ***e+1*** **non-overlapping** seeds of a read
  2. **OSS** achieves linear average case complexity, **O(e*L)**
  3. **OSS** requires a large number of seed lookups (**$O(L^2)$**)
  4. There is still room to improve the seed selection heuristics: the second best seed selection mechanism, OPS, provides **3x** more frequent seeds
- **Future work:**
  1. Develop better seed selection heuristics that approximates the optimal seeds with much fewer seed lookups and simpler algorithms
  2. Develop a fast seed lookup implementation that accommodates OSS

Acknowledgement and availability

8

- This study is supported by two NIH grants (HG006004 to C. Alkan and O. Mutlu; HG007104 to C. Kingsford) and a Marie Curie Career Integration Grant (PCIG-2011-303772) to C. Alkan

- The full manuscript of this work is available at:

Safari tech report: http://www.ece.cmu.edu/~safari/tr.html

arXiv.org: http://arxiv.org/abs/1506.08235

- The code is publically available at:

https://github.com/CMU-SAFARI/optimal-seed-solver

Optimal Seed Solver (OSS)

2

- **Challenge:** **large search space**. Seeds can start at **any position** with **any length; generate $O(L^{e+1})$ possibilities**
- **Key idea:** use **dynamic-programming** method to find the optimal seeds of **substrings** of the read
  1. Find **optimal seed positions**
  2. Find **optimal seed lengths**
- **Key recurrence relationship:** reuse the solutions of *m* seeds to calculate *m+1* seeds
- **OSS** consists of **two optimizations**:
  1. **Optimal divider cascading:** carrying over information between substrings
  2. **Early divider termination:** further reducing the search space of each substring

Optimal Divider Cascading (ODC)

4

- **OSS-DP** iterates from *1* to *e+1* seeds while in each iteration calculates the optimal solution of **all $O(e^*L^2)$ substrings**
- **Two key observations**:
  1. Only substrings that starts at the beginning of ***R*** is needed, reduce to **O(e*L) total substrings**
  2. The first optimal divider, ***P***, of a shorter substring must come first than a longer substring (Fig 2)
- **Mechanism:** Longer substrings are processed first, which **helps reduce the search space** of shorter substrings

Fig 2: In **OSS**, only substrings that starts from the beginning of R is examined. Among all substrings, the first optimal divider, |, of a shorter substring comes earlier than a longer substring, therefore, "cascading" the optimal dividers

Results

6

- **OSS** is compared against 5 previous seed selection mechanisms:
  1. Cheap K-mer Selection (CKS) mrFAST
  2. Optimal Pre-fix Selection (OPS) Hobbes
  3. Adaptive Seeds Finder (ASF) GEM
  4. Spaced Seeds (SS) PatternHunter
  5. Naïve (Fixed length, fixed placement)
- **Categorization:** length vs. placement
  1. CKS: fixed length, flexible placement
  2. OPS: fixed length, flexible placement
  3. ASF: flexible length, fixed placement
  4. SS: fixed length, fixed placement*
  5. Naïve: fixed length, fixed placement
- **Methodology:** 4 million 101-bp reads from **1000 Genome Project** (ERR240726)
  1. CKS: *12-14 bp* seeds
  2. OPS: *12-14 bp* seeds
  3. ASF: **T** = *5, 10, 100, 500, 1000* (if a read fails to produce enough seeds, ASF will roll back to CKS-*12*)
  4. SS: **pattern** = 1101001100101011111
- **Qualitative comparison:** (Table 1)
  1. Average case complexity
  2. Number of seed lookups
- **Quantitative comparison:** (Fig 4)
  1. Average frequency per seed
- **Key results:**
  1. **OSS** achieves **linear average case complexity**
  2. **OSS** provides **3x** average seed frequency reduction than the second best seed selection algorithm (OPS)

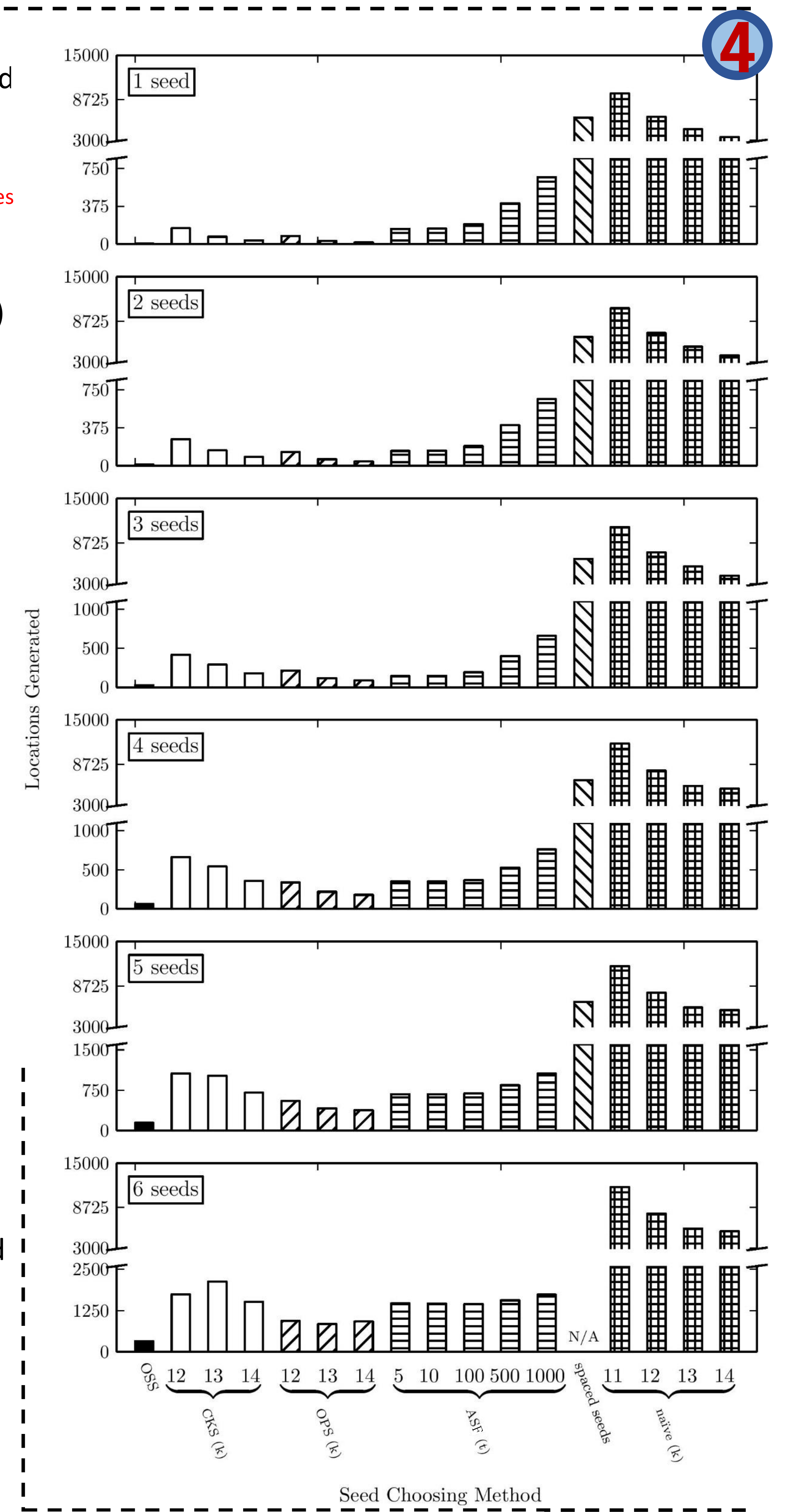

Table 1: Provides the qualitative comparison between OSS, ASF, CKS, OPS, SS and naïve. Note that OSS achieves **linear average case complexity**. In this table, x is the number of seeds while L is the length of read

	Optimal Seed Solver	ASF	CKS	OPS	Spaced seeds	naive
Empirical average case complexity	$\mathcal{O}(x \times L)$	$\mathcal{O}(x)$	$\mathcal{O}(x \times log\frac{L}{k})$	$\mathcal{O}(x \times L)$	$\mathcal{O}(x)$	$\mathcal{O}(x)$
Number of lookups	$\mathcal{O}(L^2)$	$\mathcal{O}(x)$	$\mathcal{O}(\frac{L}{k})$	$\mathcal{O}(L)$	$\mathcal{O}(x)$	$\mathcal{O}(x)$

*Spaced seeds use special patterns to balance out frequencies among seeds