Optimal Seed Solver: Optimizing Seed Selection in Read Mapping
Hongyi X1, Richard Zhu1, Sunny Nahar, John Emmons1, Gennady Pekhimenko1, Carl Kingsford1, Can Alkan2, Onur Mutlu1

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: backtracking based vs. seed-and-extend based
- NGS backtracking based mappers (e.g. bowtie2) find the best mappings fast but lose high-error mappings
- Seed-and-extend based mappers (i.e., mrfast, shrimp, RazerS3) fills all mappings but waste resources on rejecting incorrect mappings

The core dynamic-programming algorithm of OSS (OSS-DP):
- Assumption: the frequency of any single seed of the read is already known
- Baseline: enumerate all possible seed combinations, O(4^m) possibilities
- OSS: reduce the complexity to O(l4^m)
- 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, if it can 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)

Early Divider Termination (EDT):
- 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

Conclusion and future work:
- Conclusion:
  1. OSS finds the least frequent e+1 non-overlapping seeds of a read
  2. OSS achieves linear average-case complexity, O(l4^m)
  3. OSS requires a large number of seed lookups (O(l4^m))
  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:
- Develop better seed selection heuristics that approximates the optimal seeds with much fewer seed lookups and simpler algorithms
- Develop a fast seed lookup implementation that accommodates OSS

Acknowledgement and availability:
- 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.cs.cmu.edu/~safari/tr.html
- The code is publicly available at: https://github.com/CMU-SAFARI/optimal-seed-solver

Optimal Seed Solver (OSS):
- Challenge: large search space. Seeds can start at any position with any length; generate O(l4^m) 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

Problem: seed-and-extend based mappers select high frequency seeds
- Our goal: increase the efficiency of seed selection based mappers by selecting the set of least frequent e + 1 seeds with linear complexity

Optimal Divider Cascading (ODC):
- OSS-DP iterates from 1 to e+1 seeds while in each iteration calculates the optimal solution of all O(l4^m) substrings
- Two key observations:
  1. Only substrings that starts at the beginning of R is needed, reduce to O(l4^m) 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

Results:
- OSS is compared against 5 previous seed selection mechanisms:
  1. Cheap K-mer Selection (CKS)
  2. Optimal Pre-fix Selection (OPS)
  3. Adaptive Seeds Finder (ASF)
  4. Spaced Seeds (SS)
  5. Naive (Fixed length, fixed placement)
- Quantitative comparison: (Table 1)
  1. Average case complexity
  2. Number of seed lookups
  3. Qualitative comparison: (Fig 4)
  4. 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 naive. Note that OSS achieves linear average case complexity.

Optimal Seed Solver

<table>
<thead>
<tr>
<th>Number of lookups</th>
<th>Average case complexity</th>
<th>Linear case complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>OSS</td>
<td>O(l4^m)</td>
<td>O(l4^m)</td>
</tr>
<tr>
<td>ASF</td>
<td>O(l4^m)</td>
<td>O(l4^m)</td>
</tr>
<tr>
<td>CKS</td>
<td>O(l4^m)</td>
<td>O(l4^m)</td>
</tr>
<tr>
<td>OPS</td>
<td>O(l4^m)</td>
<td>O(l4^m)</td>
</tr>
<tr>
<td>SS</td>
<td>O(l4^m)</td>
<td>O(l4^m)</td>
</tr>
<tr>
<td>Naive</td>
<td>O(l4^m)</td>
<td>O(l4^m)</td>
</tr>
</tbody>
</table>

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