Optimization of Exact Algorithms for Planted $(l, d)$-Motif Search

Edward Felekey
Andrew Petersen
Samuel van Pelt
Improved Sorting

- Naïve implementation uses C++ standard library sort
  - General, all-purpose sort provided by default
- Faster sorting algorithms exist
- Customizing the sort used could improve performance
- Implemented bucket sort for faster runtime
Intersection

- Lists are merged into one large list with up to $t$ repetitions of any given candidate motif
- Reduce memory usage by intersecting lists instead of merging them
K-Split

- Divide each input sequence into \( k \) subsequences
- Run PMS1 on the first subsequence of the input sequences, then on the second, and so on
- Reduces the amount of memory used at any one time
- In this research \( k \) was set to 3
Combination

- Many of these proposed optimizations can be applied simultaneously
- Specifically, integer representation, intersection, and customized sort were combined in one implementation of the algorithm
- Both 32-bit and 64-bit versions were used
Results

- Code was written in C++
- Machines used to test runtime contained 7.7 GiB of RAM and a 2.833 GHz processor
- OpenMP was used to record execution time
- To measure memory used, the Massif tool from the Valgrind suite was run on a machine with 1GB of RAM and a 2.67 GHz processor
  - Due to memory constraints, large test cases could not be run with Massif
<table>
<thead>
<tr>
<th>ALGORITHMS</th>
<th>(9,2)</th>
<th>(11,3)</th>
<th>(13,4)</th>
<th>(15,5)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Strings</td>
<td>Time (s)</td>
<td>11.937</td>
<td>203.088</td>
<td>OOM</td>
</tr>
<tr>
<td></td>
<td>Memory (B)</td>
<td>127354880</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Integer</td>
<td>Time (s)</td>
<td>1.84891</td>
<td>29.3325</td>
<td>432.068</td>
</tr>
<tr>
<td></td>
<td>Memory (B)</td>
<td>17732744</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Intersect</td>
<td>Time (s)</td>
<td>1.43107</td>
<td>23.0912</td>
<td>343.29</td>
</tr>
<tr>
<td></td>
<td>Memory (B)</td>
<td>3257704</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Combination</td>
<td>Time (s)</td>
<td>1.39682</td>
<td>22.7992</td>
<td>341.834</td>
</tr>
<tr>
<td></td>
<td>Memory (B)</td>
<td>4543344</td>
<td></td>
<td></td>
</tr>
<tr>
<td>64-bit Combination</td>
<td>Time (s)</td>
<td>1.36052</td>
<td>22.5084</td>
<td>336.82</td>
</tr>
<tr>
<td></td>
<td>Memory (B)</td>
<td>9067200</td>
<td></td>
<td></td>
</tr>
<tr>
<td>k-Split</td>
<td>Time (s)</td>
<td>118.701</td>
<td>1825.74</td>
<td>27225.1</td>
</tr>
</tbody>
</table>
Integer Representation

- Single most significant improvement
  - More than 6 times faster than the naïve implementation for (9,2) and (11,3) challenge cases
  - Allowed for completion of (13,4), which the naïve implementation could not complete
  - Reduces space requirement by more than a factor of 7
Intersection

- Single biggest reduction in memory
  - Requires approximately 1/40 of the space of the naïve implementation for (9,2)
- Integer plus intersection runs 8 times faster, compared to 6.5 times faster with just the integer representation for (9,2)
- Only variant that finished the (15,5) case without exceeding 7.7 GiB RAM
Integer + Intersect + Custom Sort

- Introduces a tradeoff
- The custom sort algorithm runs ~3% faster on (9,2), ~1.3% faster on (11,3), and ~0.4% faster on (13,4)
- Custom sort takes up 39% more space than the Integer+Intersect version on the (9,2) challenge case
64-bit Integer+Intersect+Custom Sort

- Used twice as much memory as the 32-bit version, as expected
- Ran slightly faster than the 32-bit version
  - May be a result of differences between the 32-bit and 64-bit compilers
K-Split

- Resulted in massive slowdown – took more than 60 times longer than the naïve implementation in all cases that were completed
- Clearly did not result in a commensurate reduction in memory usage, as it was unable to complete the (15,5) case
- Not recommended to use this formulation of k-split
Use of integer representations and intersection rather than merging of lists will result in significant improvements in both runtime and memory usage.

Performance improvements extend to both 32-bit and 64-bit computers.

K-Split as presented is not recommended as an optimization.