Multi-String BWTs
A BWT containing a **string collection** instead of just a single string

- Earliest: Mantaci et al. (2005), used concatenation approach
- Bauer et al. (2011) - proposed version we will discuss today

Analogy:

- Instead of searching for a substring within a single book, search every book of a library
  - Each book has it's own text, suffix array, and end-of-text delimiter
  - Searching allows us to find how many times a substring appears and in which texts

Bioinformatics?

- Search all genomes? You could, but that's not the main application.
- Search multiple chromosomes of an organism? You should, but even that is not the killer app
Naive Construction

- Create all rotations for all strings in the collection
- Sort all rotations together (Suffix Array)
- Store the predecessor of each suffix
- Strings are “cyclic”
- The predecessor is always from the same string
- Impossible to “jump” from one string to another

<table>
<thead>
<tr>
<th>String1</th>
<th>Sorted</th>
<th>MSBWT</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACCA$</td>
<td>$ACCA</td>
<td>A</td>
</tr>
<tr>
<td>CCA$A</td>
<td>$CAAA</td>
<td>A</td>
</tr>
<tr>
<td>CA$AC</td>
<td>A$ACC</td>
<td>C</td>
</tr>
<tr>
<td>A$ACC</td>
<td>A$CAA</td>
<td>A</td>
</tr>
<tr>
<td>$ACCA</td>
<td>AAA$CA</td>
<td>A</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>String2</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>CAAA$</td>
<td>ACCA$</td>
<td>$</td>
</tr>
<tr>
<td>AAA$C</td>
<td>CA$AC</td>
<td>C</td>
</tr>
<tr>
<td>AA$CA</td>
<td>CAAA$</td>
<td>$</td>
</tr>
<tr>
<td>A$CAA</td>
<td>CCAA$</td>
<td>A</td>
</tr>
<tr>
<td>$CAA$A</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
MSBWT's FM-index

Identical Definition

- Find k-mer "CA"
- Initialize to full range ("")
- lo, hi = 0, 10
- Find occurrences of 'A'
  - lo = Offset['A'] + FMIndex[lo][A'] = 2 + 0 = 2
  - hi = Offset['A'] + FMIndex[hi][A'] = 2 + 5 = 7
- Find occurrences of "CA"
  - lo = Offset['C'] + FMIndex[lo]['C'] = 7 + 0 = 7
  - hi = Offset['C'] + FMIndex[hi]['C'] = 7 + 2 = 9
- Searching and extracting suffixes are identical to a BWT
Incremental MSBWT Construction

- A key tool missing from the BWTs toolbox—adding new strings to an existing msBWT
- You could reconstruct the suffix array of the msBWT using suffix(i, fmindex) for all i, and then insert the suffixes of the new string.
- Variant of find(); Find the insertion point of new string’s jth suffix, sj
- Add last character to msBWT
- Update the FMindex
Merging msBWTs

- BETTER YET! Rather than inserting new strings, build a BWT of the new strings and merge the new and old BWTs
- Suffixes of BTWs are already sorted
- BTWs are interleaved
- In the worse case (ties) the entire suffix must be considered, but general the longest common prefix of suffixes is smaller
- Minimal overhead
- Well suited for divide and conquer approaches (like merge sort)
- Easy to merge multiple data sets!
- Compression improves!
Merging Steps

- msBWT merging alternates between sorting and interleaving

```
<table>
<thead>
<tr>
<th>Pass #1</th>
<th>Pass #2</th>
<th>Pass #3</th>
<th>Pass #3</th>
</tr>
</thead>
<tbody>
<tr>
<td>T,0</td>
<td>S,0</td>
<td>T,0</td>
<td>S,0</td>
</tr>
<tr>
<td>G,0</td>
<td>S,0</td>
<td>G,0</td>
<td>S,0</td>
</tr>
<tr>
<td>S,0</td>
<td>S,1</td>
<td>A,1</td>
<td>S,1</td>
</tr>
<tr>
<td>A,1</td>
<td>$,1</td>
<td>A,1</td>
<td>$,1</td>
</tr>
<tr>
<td>C,0</td>
<td>A,0</td>
<td>S,0</td>
<td>A,1</td>
</tr>
<tr>
<td>S,0</td>
<td>A,0</td>
<td>T,0</td>
<td>A,1</td>
</tr>
<tr>
<td>A,0</td>
<td>A,0</td>
<td>C,0</td>
<td>A,0</td>
</tr>
<tr>
<td>A,0</td>
<td>A,1</td>
<td>G,1</td>
<td>A,1</td>
</tr>
<tr>
<td>A,1</td>
<td>A,1</td>
<td>T,1</td>
<td>A,1</td>
</tr>
<tr>
<td>A,1</td>
<td>T,1</td>
<td>A,0</td>
<td>A,0</td>
</tr>
<tr>
<td>A,1</td>
<td>T,1</td>
<td>A,0</td>
<td>A,0</td>
</tr>
<tr>
<td>A,0</td>
<td>A,0</td>
<td>T,0</td>
<td>A,0</td>
</tr>
<tr>
<td>A,0</td>
<td>A,1</td>
<td>G,1</td>
<td>A,1</td>
</tr>
<tr>
<td>A,0</td>
<td>A,1</td>
<td>G,1</td>
<td>A,1</td>
</tr>
<tr>
<td>S,1</td>
<td>T,0</td>
<td>A,0</td>
<td>T,0</td>
</tr>
<tr>
<td>A,1</td>
<td>T,0</td>
<td>A,0</td>
<td>T,0</td>
</tr>
<tr>
<td>S,1</td>
<td>T,1</td>
<td>A,1</td>
<td>T,1</td>
</tr>
<tr>
<td>A,1</td>
<td>T,1</td>
<td>A,1</td>
<td>T,1</td>
</tr>
</tbody>
</table>
```

Sort by base, Select from BWT, Sort by base, Select from BWT, Sort by base, Select from BWT, Done!
```python
def mergeBWT(bwt1, bwt2):
    interleave = [(c, 0) for c in bwt1] + [(c, 1) for c in bwt2]
    passes = min(len(bwt1), len(bwt2))
    for p in xrange(passes):
        i, j = 0, 0
        nextInterleave = []
        for c, k in sorted(interleave, key=lambda x: x[0]):
            if (k == 0):
                b = bwt1[i]
                i += 1
            else:
                b = bwt2[j]
                j += 1
            nextInterleave.append((b, k))
            if (nextInterleave == interleave):
                break
        interleave = nextInterleave
    return ''.join(['c' for c, k in interleave])

bwt1 = "TG$TC$AAAA"
bwt2 = "AAGT$GTA$AS"
bwt12 = mergeBWT(bwt1, bwt2)
print bwt12
FM, Offset = FMIndex(bwt12)
for i in xrange(len(bwt12)):
    print "%2d: %s" % (i, recoverSuffix(i, bwt12, FM, Offset))
```

TGAAG$TGC$AAA$AAA$

0: $ACAT$
1: $ATAG$
2: $GAGA$
3: $TATA$
4: $A$GAG
5: $A$TAT
6: ACAT$
7: AG$AT$
8: AG$AG$
MSBWT Applications

- Instead of building a BWT of a reference genome, build a MSBWT of every sequenced reads
- Arbitrary exact-match k-mer queries
- O(k) time
- Enables fast searches/counting
- Recover an arbitrary read of length L from MSBWT
- O(L) time
- Enables extraction of user-selected reads
Compression of high-throughput sequencing

- Using Run-length encoding again
- Reasons we expect compression:
  - True genomic repeats: gene families, long repeats, etc.
  - Over-sampling: 30x coverage means we expect 30 copies of every k-mer pattern
  - Sequencing errors may break up runs
  - Technical errors may cause biases for or against a particular pattern
- Real Mouse DNA-seq:
  - $368654191 \times 151 \times 2 = \sim 112$ Giga-bases
  - Compresses to $\sim 15.3$ GB using RLE (1.09 bits/base)
- Real Mouse RNA-seq:
  - $\sim 8.9$ Giga-bases
  - $\sim 1.2$ GB using RLE (1.05 bits/base)
K-mer Search & Extraction

Basic Use:


- Search for all reads with a given k-mer
- Extract all reads with that k-mer and its reverse-complement
- Build a consensus
Reference-based Searches

- Given a reference genome and region of that genome
- Split reference into k-mers
- Count the abundance of each k-mer and plot
- Fast - $O(k)$ time per k-mer
- Similar to a post-alignment pileup
Reference Correction

CAST/EiJ DNA-seq for annotated gene Igf2
Summary

- Burrows-Wheeler Transform
  - Permutation of characters that represents a suffix array
  - Run-length encoded for compression
- FM-index
  - Derived from BWT
  - Exploits LF-mapping property
  - $O(k)$ search time for arbitrary k-mer, independent of BWT's size
  - Used in many fast aligners
- MSBWT
  - Applies to string collections
  - Enables database-like access to reads via k-mer searches
Next Time

Hidden Markov Models