

Extensions to CSE Discovery

Barna Zajzon

Practical Course in Bioinformatics
RWTH Aachen

June 29, 2015

Outline

- ① Goals
- ② Implementation Details
- ③ Results
- ④ Summary

Goals

- ① Initial Improvements
- ② Indel Detection
- ③ Higher Order Errors
- ④ Hashing - Memory Usage

Initial Improvements

Bugs

- Index error on forward strands → motif inconsistency:

```
qgram = genome [i : i+q]          # pos 10-19  
... genome_annotate[0][i + q]    # trying pos 20
```

Changed to

```
genome_annotate[0][i + q - 1]    # trying pos 19
```

Initial Improvements

Bugs

- Index error on forward strands → motif inconsistency:

```
qgram = genome [i : i+q]          # pos 10-19
... genome_annotate[0][i + q]    # trying pos 20
```

Changed to

```
genome_annotate[0][i + q - 1]    # trying pos 19
```

- Last motif in genome was missed:

```
for i in range(len(genome) - q):
```

Changed to

```
for i in range(len(genome) - q + 1):
```

Initial Improvements

Bugs

- Error while merging (F + R strands) annotated qgrams
→ motifs only on REV strand missed:

```
for qgram in qgram_last:
    qgram_rev = reverse(qgram)
    if qgram_rev in qgram_first:
        ...
```

Solution

- pack motifs on F-strand in result
- iterate over motifs on R-strand
- merge or add if not present

Initial Improvements

Speedup

- Use SciPy's *chi squared test* for large tables
- Keep R's *Fisher's test*
- Intermediate counts for motifs instead of regex

Indel Detection

- Create similar contingency tables during genome annotate

	8	9	10	-	11
	A	G	C	x	T
F-strand	A	G	-	C	T
F-strand	A	G	-	-	G
R-strand	A	C	A	C	T
R-strand	A	A	-	-	T
F-strand	A	G	-	A	T

Deletion Table(10)	Match	Mismatch	Insertion Table(10)	Match	Mismatch
F-strand	3	0	F-strand	2	1
R-strand	1	1	R-strand	1	1

Indel Detection

- Create similar contingency tables during genome annotate

	8	9	10	-	11
	A	G	C	x	T
F-strand	A	G	-	C	T
F-strand	A	G	-	-	G
R-strand	A	C	A	C	T
R-strand	A	A	-	-	T
F-strand	A	G	-	A	T

Deletion Table(10)	Match	Mismatch	Insertion Table(10)	Match	Mismatch
F-strand	3	0	F-strand	2	1
R-strand	1	1	R-strand	1	1

- Parse cigars for *I* and *D*
- Tripled annotation dictionary (qgram) and array (genome)

Higher Order Errors

- Flexible offset during qgram annotation
- Arbitrary position after motif (before on R-strand)
- Drawback: one run = one position

Efficient Hashing

$$\overbrace{A \textcolor{red}{C} T \textcolor{red}{C}}^{k=4} \rightarrow \overbrace{\underbrace{0 \ 0}_{2 \cdot k - 1 \text{ (key)}} \underbrace{\textcolor{red}{0} \textcolor{red}{1} \textcolor{red}{1} \textcolor{red}{1} \textcolor{red}{0} \textcolor{red}{1}}_{l \text{ (position)}}}_{2 \cdot k = 8} = m$$

- Size of hash table: $M = 2^l$
- Binary, $2k \times 2k$ invertible matrix A

Efficient Hashing

$$\overbrace{A \textcolor{red}{C} T \textcolor{red}{C}}^{k=4} \rightarrow \underbrace{00}_{2^*k-l \text{ (key)}} \underbrace{\textcolor{red}{0} \textcolor{red}{1} \textcolor{red}{1} \textcolor{red}{1} \textcolor{red}{0} \textcolor{red}{1}}_{l \text{ (position)}}^{2^*k=8} = m$$

- Size of hash table: $M = 2^l$
- Binary, $2k \times 2k$ invertible matrix A

$$f(m) = A * m = 10\textcolor{blue}{110010} \quad (1)$$

$$\text{hash}(m) = f(m) \% M = \textcolor{blue}{110010} \quad (2)$$

$$\text{pos}(m, i) = (\text{hash}(m) + \text{reprobe}(i)) \% M \quad (3)$$

Efficient Hashing

$$\overbrace{A \text{CTC}}^{k=4} \rightarrow \underbrace{00}_{2*k-l \text{ (key)}} \underbrace{011101}_{l \text{ (position)}}^{2*k=8} = m$$

- Size of hash table: $M = 2^l$
- Binary, $2k \times 2k$ invertible matrix A

$$f(m) = A * m = 10110010 \quad (1)$$

$$hash(m) = f(m) \% M = 110010 \quad (2)$$

$$pos(m, i) = (hash(m) + reprobe(i)) \% M \quad (3)$$

- $i + 1$ is concatenated to the higher $2 * k - l$ bits

Efficient Hashing

$$f(m) = A * m = \textcolor{red}{10110010} \quad (4)$$

$$\textit{hash}(m) = f(m) \% M = \textcolor{blue}{110010} \quad (5)$$

$$\textit{pos}(m, i) = (\textit{hash}(m) + \textit{reprobe}(i)) \% M \quad (6)$$

Efficient Hashing

$$f(m) = A * m = \textcolor{red}{10}\textcolor{blue}{110010} \quad (4)$$

$$\text{hash}(m) = f(m) \% M = \textcolor{blue}{110010} \quad (5)$$

$$\text{pos}(m, i) = (\text{hash}(m) + \text{reprobe}(i)) \% M \quad (6)$$

hashtable pos	49	50 = $\textcolor{blue}{110010}$	51
key		$\textcolor{red}{1}\textcolor{red}{10}$	
value		[fm, rm, fmm, rmm]	

Efficient Hashing

Problems / Tricks

- With “N”, $3 * k$ needed for encoding
- Numpy arrays: 1 byte * #bits
- Integers & operations on bits instead “bitvectors”
- One list as hash table entry
- Matrix A actually not needed?
- Memory improvement, but terrible speed

Results

CSE Discovery

	Original		All improvements	
bsubtilis	Time (s)	Memory Max (MB)	Time (s)	Memory Max (MB)
Q=4, N=2	421.289	223	235.481	231
Q=8, N=2	4083.274	251	367.882	265

	Original		All improvements	
bordetella	Time (s)	Memory Max (MB)	Time (s)	Memory Max (MB)
Q=4, N=2	132.779	217	58.059	225
Q=8, N=2	870.690	242	357.841	254

Results

Hashing

bsubtilis	Dictionary		Key encoding		C++ with Python	
	Time	Memory	Time	Memory	Time	Memory
Q=10, N=0	4s	335	60s	309	16s	125
Q=10, N=1	14s	664	159s	535	x	x
Q=10, N=2	32s	1.257	302s	914	x	x

Summary

- Major and initial goals achieved
- Hashing only proof-of-concept
- Better possible (merge data structures, parallelization)
- More *testing* required, especially for indels
- Is k-mer hashing worth it as pure Python?