Memory Efficient Minimum Substring Partitioning

Yang Li, Pegah Kamousi, Fangqiu Han, Shengqi Yang, Xifeng Yan, Subhash Suri

Dept. of Computer Science, UC Santa Barbara
Motivation - Challenges

• De Bruijn graph based sequence assembly

• Building de Bruijn graph is both time intensive and memory consuming
Motivation - Existing solutions

- In-memory solution
  - Smiley face: Small running time
  - Sad face: Huge memory footprint

- Classic disk-based approach
  - Smiley face: Small memory footprint
  - Sad face: Huge disk space consumption
  - Sad face: Very large running time
Objectives

• De Bruijn graph construction
  - with small running time
  - with small memory footprint
  - with small disk space consumption

• Minimum Substring Partitioning

A disk-based method with a SMART partitioning strategy 😊
Outline

- Backgrounds
- Minimum Substring Partitioning (MSP)
- MSP-based de Bruijn graph construction
- Experiments
- Conclusions
De Bruijn graph

- \( G_k = (V,E) \)
- \( V = \) All unique \( k \)-mers (length-\( k \) substrings)
- \( E = \) Directed edges between consecutive \( k \)-mers
  - consecutive \( k \)-mers overlap by \( k-1 \) symbols
- Human genome: >3B nodes, >10B edges

**Reads**

<table>
<thead>
<tr>
<th>Reads</th>
<th>de Bruijn Graph (k=3)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACCAACGTTG</td>
<td>ACC → CCA</td>
</tr>
<tr>
<td>AGCAACTCGT</td>
<td>AGC → GCA → CAA → AAC → ACG → CGT → GTT → TTG</td>
</tr>
</tbody>
</table>
• Goal: deduplicate k-mers

• Horizontal Partition (**H-Partition**)  
  1. partition reads $S$ horizontally into disjoint subsets $S_i$  
  2. for each $S_i$, build a hash table $H_i$ of k-mers in memory  
  3. output a sorted copy $H_i$ to disk  
  4. merge all such sorted hash tables

**Pro:** partitioning is simple and straightforward

**Con:** merging is very expensive (time consuming)
  
  • same k-mer may appear in different partitions
• Goal: deduplicate k-mers

• Bucket Partition (\textit{B-Partition})
  1. partition(hash) all k-mers from S into disjoint subsets \( K_i \)
  2. for each \( K_i \), build a hash table \( H_i \) of k-mers in memory
  3. output \( H_i \) (no need to sort) to disk
  4. merge all such hash tables

\begin{itemize}
  \item \green{Pro: merging is simple and straightforward}
  \item \red{Con: partitioning is very expensive (time consuming)}
    \begin{itemize}
      \item k-mer set size is much larger than sequence set size
      \item huge I/O costs and disk space occupations
    \end{itemize}
\end{itemize}
Minimum Substring Partitioning

**Intuition**
- if several adjacent k-mers are distributed to the same partition, we can compress them to reduce I/O costs

**Observation**
- since two adjacent k-mers overlap with length k-1 substring, the chance for them to have the same minimum p-substring ($p < k$) could be very high.

**Idea**
- partition k-mers w.r.t minimum substring
Minimum Substring Partitioning

Given a string $s = s_1 s_2 \ldots s_m$, $p \leq k \leq m$, minimum substring partitioning breaks $s$ to substrings with maximum length $\{s[i, j]|i + k - 1 \leq j, 1 \leq i, j \leq m\}$, s.t., all $k$-mers in $s[i, j]$ share the same minimum $p$-substring. $s[i, j]$ is also called super $k$-mer.

Instead of writing 7 length-16 $k$-mers to disk, now you can just output a length-18 and a length-19 super $k$-mers! Great save!
Minimum Substring Partitioning - Theorems

- We employ a *random string model* to derive theorems.

- **Total Partition Size**

  **Theorem 1**
  
  *In a random string model, the total partition size is $O(pn)$ or $\Theta(n)$.*

- **Largest Partition Capacity**

  **Theorem 2**
  
  *In a random string model, the maximum percentage of distinct k-mers covered by one $p$-substring is bounded by $3k/(4^p + 1)$, when $p \geq 2$.*

Critical for running time & disk space usage:

Critical for peak memory:
MSP-based de Bruijn graph construction

**Partitioning**
A simple yet efficient scan algorithm for MSP

**Mapping**
Assign ID to each k-mer and generate ID replacement tables

**Merging**
Merge ID replacement tables to produce a disk-based de Bruijn graph

8/28/13
UCSB
Experiments - Setup

• **Datasets**

<table>
<thead>
<tr>
<th></th>
<th>Cladonema</th>
<th>Bee</th>
<th>Fish</th>
<th>Bird</th>
</tr>
</thead>
<tbody>
<tr>
<td>Size (GB)</td>
<td>258.7</td>
<td>93.8</td>
<td>137.5</td>
<td>106.8</td>
</tr>
<tr>
<td>Avg Read Length (bp)</td>
<td>101</td>
<td>124</td>
<td>101</td>
<td>150</td>
</tr>
<tr>
<td># of Reads (million)</td>
<td>894</td>
<td>303</td>
<td>598</td>
<td>323</td>
</tr>
</tbody>
</table>

• **Environment**
  • a server with 2.40GHz Intel Xeon CPU and 512GB RAM

• **Evaluation criteria**
  • Peak memory
  • Running time
  • Disk space usage

*The smaller, the better!*
Experiments - Efficiency

An order of magnitude reduction of memory usage

(a) Peak Memory
(b) Running Time
Experiments - Effectiveness

Reduce disk space usage by more than 10 times

(a) Max Disk Space Usage

(b) Running Time
Experiments - Scalability

Linear scalability

(a) Peak Memory

(b) Running Time

Memory Consumption (GB)

Running Time (x1000 seconds)

Velvet
SOAPdenovo
MSP
Experiments - Properties (1)

Peak memory decreases significantly as $p$ increases

Partition size and running time slightly increase as $p$ increases

(a) **Peak Memory**

(b) **Total Partition Size**

(c) **Running Time**
Experiments - Properties (2)

Peek memory increases slowly as $k$ increases

Partition size and running time decrease as $k$ increases
Conclusions

• Minimum Substring Partitioning
  • with small running time
  • with small memory footprint
  • with small disk space consumption

• Project Homepage
  • http://grafia.cs.ucsb.edu/msp
Remaining Challenges

Sequence Assembly

Construct de Bruijn graph

Load graph & generate sequence
Thanks!