Gap Filling as Exact Path Length Problem
RECOMB 2015

Leena Salmela\textsuperscript{1}  Kristoffer Sahlin\textsuperscript{2}
Veli Mäkinen\textsuperscript{1}  Alexandru I. Tomescu\textsuperscript{1}

\textsuperscript{1}University of Helsinki
\textsuperscript{2}KTH Royal Institute of Technology

April 12th, 2015
Gap filling

- Gap filling is the last phase in genome assembly
- Input: Scaffolds (=linearly ordered contigs) and reads
- Output: Scaffolds where gaps between contigs have been filled
Previous work

- Gap filling module in many popular assemblers:
  - Allpaths-LG
  - ABYSS
  - EULER
  - ...

- Standalone gap filling tools:
  - SOAPdenovo’s GapCloser
  - GapFiller (Boetzer & Pirovano 2012)

- General idea:
  - Identify reads potentially filling the gap
  - Local assembly
Our contribution

- Problem formulation as \textit{Exact Path Length problem}
- Gap Filling is \textit{NP-complete}
- \textit{Pseudopolynomial algorithm} for Gap Filling
- Implementation of the algorithm in a tool called \textit{Gap2Seq}
Gap filling: Problem definition

Given

- an (overlap or de Bruijn) graph $G = (V, E)$ of the whole read set
- a cost function $c : E \mapsto \mathbb{Z}_+$
- two vertices $s$ and $t$ representing the flanks of the contigs
- estimate of the gap length $[d', d]$

find for all $x \in [d', d]$ the number of paths $P = v_1, v_2, \ldots, v_k$ such that

$$cost(P) = \sum_{i=1}^{k-1} c(v_i, v_{i+1}) = x.$$
Gap filling: Problem definition

Given

- an (overlap or de Bruijn) graph $G = (V, E)$ of the whole read set
- a cost function $c : E \mapsto \mathbb{Z}_+$
- two vertices $s$ and $t$ representing the flanks of the contigs
- estimate of the gap length $[d', d]$

find for all $x \in [d', d]$ the number of paths $P = v_1, v_2, \ldots, v_k$ such that

$$\text{cost}(P) = \sum_{i=1}^{k-1} c(v_i, v_{i+1}) = x.$$

NP-complete
Dynamic programming algorithm

- For each $v \in V(G)$ and $\ell \in [0, d]$ define:

$$a(v, \ell) = \text{number of } s \rightarrow v \text{ paths of cost } \ell$$
Dynamic programming algorithm

- For each $v \in V(G)$ and $\ell \in [0, d]$ define:
  \[
a(v, \ell) = \text{number of } s - v \text{ paths of cost } \ell
  \]

Initialize: $a(s, 0) = 1$
Dynamic programming algorithm

- For each $v \in V(G)$ and $\ell \in [0, d]$ define:
  
  $$a(v, \ell) = \text{number of } s - v \text{ paths of cost } \ell$$

- Recurrence: $a(v, \ell) = \sum_{u \in N^-(v)} a(u, \ell - c(u, v))$
  
  where $N^-(v)$ is the set of in-neighbors of $v$

- Initial conditions:
  
  $$a(s, 0) = 1$$
  $$a(v, 1) = \sum_{u \in N^-(v)} a(u, 0) = 1$$
Dynamic programming algorithm

- For each \( v \in V(G) \) and \( \ell \in [0, d] \) define:

\[
a(v, \ell) = \text{number of } s - v \text{ paths of cost } \ell
\]

- Recurrence: \( a(v, \ell) = \sum_{u \in N^{-}(v)} a(u, \ell - c(u, v)) \)

where \( N^{-}(v) \) is the set of in-neighbors of \( v \)

\[
a(s, 0) = 1, \quad a(v, 1) = 1, \quad a(v, 1) = 1, \quad a(v, 2) = \sum_{u \in N^{-}(v)} a(u, 1) = 2
\]
For each $v \in V(G)$ and $\ell \in [0, d]$ define:

$$a(v, \ell) = \text{number of } s - v \text{ paths of cost } \ell$$

Recurrence: $a(v, \ell) = \sum_{u \in N^-(v)} a(u, \ell - c(u, v))$

where $N^-(v)$ is the set of in-neighbors of $v$
Dynamic programming algorithm

- For each $v \in V(G)$ and $\ell \in [0, d]$ define:

  $$a(v, \ell) = \text{number of } s - v \text{ paths of cost } \ell$$

- Recurrence:  
  $$a(v, \ell) = \sum_{u \in N^-(v)} a(u, \ell - c(u, v))$$ 
  where $N^-(v)$ is the set of in-neighbors of $v$}

**Pseudopolynomial algorithm running in $O(dm)$ time**

($d$: length of gap, $m$: number of arcs)
Choosing the path

- If there are several paths:
  1. Choose the one closest to \((d' + d)/2\)
  2. If several such paths, choose one at random.

- Backtracing in the DP matrix gives the path
Implementation: Gap2Seq

- Build a de Bruijn graph of the reads
  - We use GATB for efficient implementation of the DBG
- Use a hash table to link reachable vertices to their DP table rows
- DP table rows are sparse
  - List only non-zero entries
- $k$-mers flanking gaps can have errors
  - Allow paths to start/end at up to $e$ flanking $k$-mers
- Parallelisation on the scaffold level
- Limit the memory usage of the DP table
  - Abandon search on a gap if limit exceeded
Experimental results: S. aureus GAGE data

Experiments run on all 8 GAGE assemblies.

We show aggregates over all assemblies.
Experimental results: *S. aureus* GAGE data

- Experiments run on all 8 GAGE assemblies.
- We show aggregates over all assemblies.
Further work

- Scaling to larger genomes
- Improving runtime and memory usage
  - *Meet-in-the-middle*: start the search from both flanks of the gap
Thanks!
Questions?

http://www.cs.helsinki.fi/u/lmsalmel/Gap2Seq/