Space-efficient merging of succinct de Bruijn graphs

Giovanni Manzini
University Eastern Piedmont, Alessandria
IIT-CNR, Pisa, Italy

Joint work with
Lavinia Egidi and Felipe Louza

SPIRE
Segovia, October 8th 2019

Outline

1. Introduction to de Bruijn graphs
2. Succinct representation of de Bruijn graphs
3. Merging succinct de Bruijn graphs

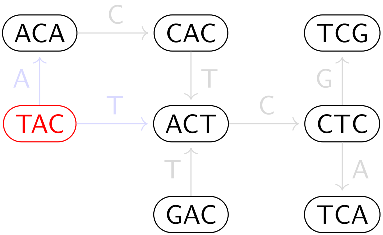
de Bruijn graphs (dBGs)

Definitions:

- Given a collection of strings $\mathcal{S}$, a *de Bruijn* graph of order k is a directed graph containing:
  - a node v for every **unique** k**-mer** $v[1]\ldots v[k]$ in $\mathcal{S}$.
  - an edge (u, v) with label $v[k]$ if there is a $(k+1)$**-mer** $u[1]\ldots u[k]v[k]$ in $\mathcal{S}$.

Example:

- $\mathcal{S} = \{$TACACT, TACTCA, GACTCG$\}$



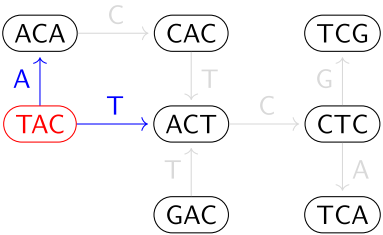
de Bruijn graphs (dBGs)

Definitions:

- Given a collection of strings $\mathcal{S}$, a *de Bruijn* graph of order k is a directed graph containing:
  - a node v for every **unique** k**-mer** $v[1]...v[k]$ in $\mathcal{S}$.
  - an edge (u, v) with label $v[k]$ if there is a $(k+1)$**-mer** $u[1]...u[k]v[k]$ in $\mathcal{S}$.

Example:

- $\mathcal{S} = \{$TACACT, TACTCA, GACTCG$\}$



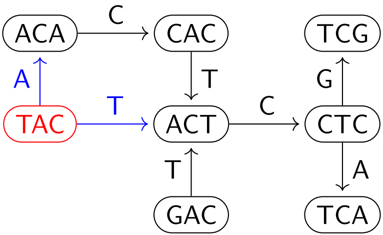
de Bruijn graphs (dBGs)

Definitions:

- Given a collection of strings $\mathcal{S}$, a *de Bruijn* graph of order k is a directed graph containing:
  - a node v for every **unique** k**-mer** $v[1]...v[k]$ in $\mathcal{S}$.
  - an edge (u, v) with label $v[k]$ if there is a $(k+1)$**-mer** $u[1]...u[k]v[k]$ in $\mathcal{S}$.

Example:

- $\mathcal{S} = \{$TACACT, TACTCA, GACTCG$\}$



Comments

- Sometimes colors are assigned to (groups of) strings so edges are colored
- At the core it is a data structure supporting existential queries on k-mers, and the retrieval of "overlapping" k-mers
- Compressed suffix trees and bidirectional FM-indices support more operations using slightly more space

Outline

1. Introduction to de Bruijn graphs

2. Succinct representation of de Bruijn graphs

3. Merging succinct de Bruijn graphs

Succinct representation of dBGs:

BOSS*:

- Bowe *et al.* in [WABI 2012] introduced a succinct representation for dBGs using space $m(\log \sigma + 2) + o(m)$ bits, where $m = |E|$.
- Each edge is represented by its symbol plus 2 bits
- Additional rank/select data structures to support efficient navigation
- Not the only known succinct representation of dBGs

Example:

- $\mathcal{S} = \{$ TACACT, TACTCA, GACTCG$\}$

*for the authors' initials

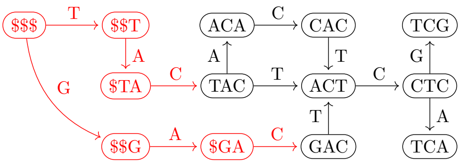
Succinct representation of dBGs:

BOSS*:

- Bowe *et al.* in [WABI 2012] introduced a succinct representation for dBGs using space $m(\log \sigma + 2) + o(m)$ bits, where $m = |E|$.
- Each edge is represented by its symbol plus 2 bits
- Additional rank/select data structures to support efficient navigation
- Not the only known succinct representation of dBGs

Example:

- $\mathcal{S} = \{$\$\$\$TACACT, \$\$\$TACTCA, \$\$\$GACTCG$\}$



---

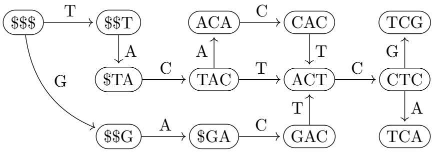
*For convenience, we add k copies of a symbol \$ at the beginning of each string s_i.

Succinct representation of dBGs:

BOSS:

- Nodes v_i are sorted by their **reversed labels** $\overleftarrow{v_i}$, and we list the symbols in the outgoing edges obtaining array W
- In array last we mark the position of the **last outgoing edge** of each node.
- In array W we mark (with a $^\bullet$) the symbols associated to the **2nd, 3rd, . . .** edge entering in a node.

Nodes	W
\$ \$ \$	G
	T
A C A	C
T C A	#
\$ G A	C
\$ T A	C
C A C	T
G A C	T
T A C	A
	T
C T C	A
	G
\$ \$ G	A
T C G	#
\$ \$ T	A
A C T	C

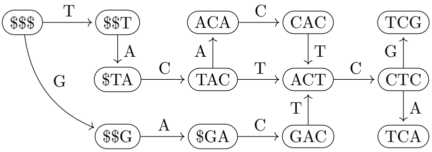


Succinct representation of dBGs:

BOSS:

- Nodes v_i are sorted by their **reversed labels** $\overleftarrow{v_i}$, and we list the symbols in the outgoing edges obtaining array W
- In array last we mark the position of the **last outgoing edge** of each node.
- In array W we mark (with a $^{\bullet}$) the symbols associated to the **2nd, 3rd, . . .** edge entering in a node.

last	Nodes	W
0	$ $ $	G
1	.	T
1	A C A	C
1	T C A	#
1	$ G A	C
1	$ T A	C
1	C A C	T
1	G A C	T
0	T A C	A
1	.	T
0	C T C	A
1	.	G
1	$ $ G	A
1	T C G	#
1	$ $ T	A
1	A C T	C

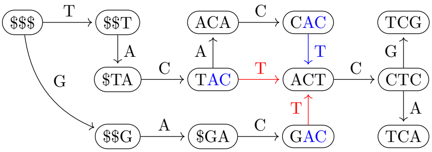


Succinct representation of dBGs:

BOSS:

- Nodes v_i are sorted by their **reversed labels** $\overleftarrow{v_i}$, and we list the symbols in the outgoing edges obtaining array W
- In array last we mark the position of the **last outgoing edge** of each node.
- In array W we mark (with a $^\bullet$) the symbols associated to the **2nd, 3rd, ...** edge entering in a node.

last	Nodes	W
0	$ $ $	G
1	.	T
1	A C A	C
1	T C A	#
1	$ G A	C
1	$ T A	C
1	C A C	T
1	G A C	T•
0	T A C	A
1	.	T•
0	C T C	A
1	.	G
1	$ $ G	A
1	T C G	#
1	$ $ T	A
1	A C T	C

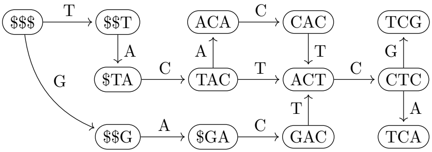


Succinct representation of dBGs:

BOSS:

- Nodes v_i are sorted by their **reversed labels** $\overleftarrow{v_i}$, and we list the symbols in the outgoing edges obtaining array W
- In array last we mark the position of the **last outgoing edge** of each node.
- In array W we mark (with a $^\bullet$) the symbols associated to the **2nd, 3rd, . . .** edge entering in a node.

last	W
0	G
1	T
1	C
1	#
1	C
1	C
1	T
1	T$^\bullet$
0	A
1	T$^\bullet$
0	A
1	G
1	A
1	#
1	A
1	C



Outline

1. Introduction to de Bruijn graphs

2. Succinct representation of de Bruijn graphs

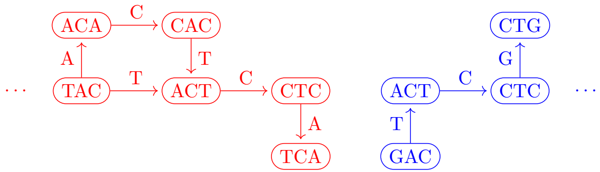
3. Merging succinct de Bruijn graphs

Merging de Bruijn graphs

- Suppose we are given the BOSS representation of two de Bruijn graphs G_0 and G_1 for the collections of strings $\mathcal{C}_0$ and $\mathcal{C}_1$
- We want to compute the BOSS for $\mathcal{C}_{01} = \mathcal{C}_0 \cup \mathcal{C}_1$ directly, that is, without decoding G_0 and G_1.
- Working space is a major issue, since it limits the size of the largest graph we can build

Example:

$\mathcal{S}_1 = \{\$\$\$\text{TACACT}, \$\$\$\text{TACTCA}\} \cup \{\$\$\$\text{GACTCG}\}$



Merging de Bruijn Graphs

Merging BOSS representations

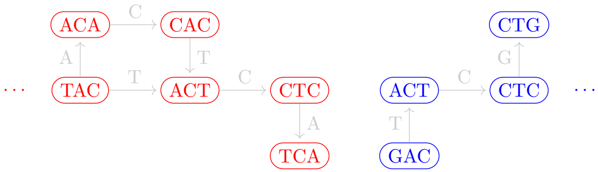
- Tasks:
  1. Merge the nodes in G_0 and G_1 according the order of their k-mers,

$$\overleftarrow{v_1} \prec \cdots \prec \overleftarrow{v_{n_0}} \qquad \text{and} \qquad \overleftarrow{w_1} \prec \cdots \prec \overleftarrow{w_{n_1}}$$

  2. Recognize when two nodes in G_0 and G_1 refer to the same k-mer, and

$$\overleftarrow{v_i} \ ?= \overleftarrow{w_j}$$

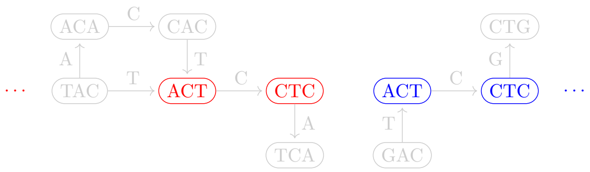
  3. Properly merge and update W and last.



Merging de Bruijn Graphs

Merging BOSS representations

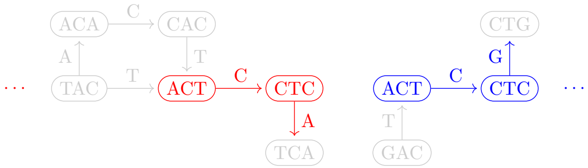
- Tasks:
  1. Merge the nodes in G_0 and G_1 according the order of their k-mers,
  $$\overleftarrow{v_1} \prec \cdots \prec \overleftarrow{v_{n_0}} \qquad \text{and} \qquad \overleftarrow{w_1} \prec \cdots \prec \overleftarrow{w_{n_1}}$$
  2. Recognize when two nodes in G_0 and G_1 refer to the same k-mer, and
  $$\overleftarrow{v_i} \; ?= \overleftarrow{w_j}$$
  3. Properly merge and update W and last.



Merging de Bruijn Graphs

Merging BOSS representations

- Tasks:
  1. Merge the nodes in G_0 and G_1 according the order of their k-mers,
     $$\overleftarrow{v_1} \prec \cdots \prec \overleftarrow{v_{n_0}} \qquad \text{and} \qquad \overleftarrow{w_1} \prec \cdots \prec \overleftarrow{w_{n_1}}$$
  2. Recognize when two nodes in G_0 and G_1 refer to the same k-mer, and
     $$\overleftarrow{v_i} \; ?\!= \overleftarrow{w_j}$$
  3. Properly merge and update W and last.



Observation:

- Order preserving bijection between the symbols in *W* and *Nodes*...
- ... excluding those marked with •
- We get the last symbol of each node (first in the ordering)

last	Nodes	W
0	? ? $	G
1	.	T
1	? ? A	C
1	? ? A	$
1	? ? A	C
1	? ? A	C
1	? ? C	T
1	? ? C	T•
0	? ? C	A
1	.	T•
0	? ? C	A
1	.	G
1	? ? G	A
1	? ? G	$
1	? ? T	A
1	? ? T	C

Observation:

- Order preserving bijection between the symbols in *W* and *Nodes*...
- ... excluding those marked with $^\bullet$
- We get the last symbol of each node (first in the ordering)

last	Nodes	W
0	? ? $	G
1	.	T
1	? ? A	C
1	? ? A	$
1	? ? A	C
1	? ? A	C
1	? ? C	T
1	? ? C	T$^\bullet$
0	? ? C	A
1	.	T$^\bullet$
0	? ? C	A
1	.	G
1	? ? G	A
1	? ? G	$
1	? ? T	A
1	? ? T	C

Observation:

- Order preserving bijection between the symbols in W and $Nodes$...
- ... excluding those marked with $^{\bullet}$
- We get the last symbol of each node (first in the ordering)

last	Nodes	W
0	? ? \$	G
1	.	T
1	? ? A	C
1	? ? A	\$
1	? ? A	C
1	? ? A	C
1	? ? C	T
1	? ? C	$T^{\bullet}$
0	? ? C	A
1	.	$T^{\bullet}$
0	? ? C	A
1	.	G
1	? ? G	A
1	? ? G	\$
1	? ? T	A
1	? ? T	C

Previous approach (Muggli et al. Bioinformatics '19)

- Recover labels columnwise right to left.
- Do this for both dbGs and simultaneously merge nodes
- Working space: $2(|V|\log\sigma + |E| + |V|)$ bits

last	Nodes	W
0	? ? $	G
1	.	T
1	? ? A	C
1	? ? A	$
1	? ? A	C
1	? ? A	C
1	? ? C	T
1	? ? C	T•
0	? ? C	A
1	.	T•
0	? ? C	A
1	.	G
1	? ? G	A
1	? ? G	$
1	? ? T	A
1	? ? T	C

Previous approach (Muggli et al. Bioinformatics ’19)

- Recover labels columnwise right to left.
- Do this for both dbGs and simultaneously merge nodes
- Working space: $2(|V| \log \sigma + |E| + |V|)$ bits

last	Nodes	W
0	? $ $	G
1	.	T
1	? C A	C
1	? C A	$
1	? G A	C
1	? T A	C
1	? A C	T
1	? A C	T•
0	? A C	A
1	.	T•
0	? T C	A
1	.	G
1	? $ G	A
1	? C G	$
1	? $ T	A
1	? C T	C

Previous approach (Muggli et al. Bioinformatics '19)

- Recover labels columnwise right to left.
- Do this for both dbGs and simultaneously merge nodes
- Working space: $2(|V|\log\sigma + |E| + |V|)$ bits

last	Nodes	W
0	\$ \$?	G
1	.	T
1	A C ?	C
1	T C ?	\$
1	\$ G ?	C
1	\$ T ?	C
1	C A ?	T
1	G A ?	T•
0	T A ?	A
1	.	T•
0	C T ?	A
1	.	G
1	\$ \$?	A
1	T C ?	\$
1	\$ \$?	A
1	A C ?	C

Our approach: induced sorting

- Merge nodes according to the rightmost h symbols for $h = 1, 2, \dots, k$
- At each iteration edge labels are used to refine the merging
- Inspired by Holt and McMillan [Bioinformatics 2014, ACM-BCB 2014]

last	Nodes	W
0	$ $ $ $	G
1	.	T
1	? ? C A	A
0	? ? C A	C
1	.	T
1	$ G A	C
1	? $ T A	C•
0	? ? A C	A
1	.	T•
0	? ? A C	A
1	.	T•
0	? ? T C	A
1	.	C
1	? ? $ G	A
1	? ? C G	#
1	? ? $ T	A
1	? ? C T	C

Our approach: induced sorting

- Merge nodes according to the rightmost h symbols for $h = 1, 2, \ldots, k$
- At each iteration edge labels are used to refine the merging
- Inspired by Holt and McMillan [Bioinformatics 2014, ACM-BCB 2014]

last	Nodes	W		Nodes
0	\$ \$ \$ \$	G		\$ \$ \$ \$
1	.	T		.
1	? ? C A	A		? T C A
0	? ? C A	C		? ? C A
1	.	T		.
1	? G A	C		? G A
1	? ? T A	C•		? C A C
0	? ? A C	A	→	? T C C
1	.	T•		.
0	? ? A C	A		? ? A C
1	.	T•		.
0	? ? T C	A		? ? T C
1	.	C		.
1	? ? \$ G	A		? ? \$ G
1	? ? C G	#		? ? C G
1	? ? \$ T	A		? ? \$ T
1	? ? A T	C		? C A T

Our approach: induced sorting

- Merge nodes according to the rightmost h symbols for $h = 1, 2, \ldots, k$
- At each iteration edge labels are used to refine the merging
- Inspired by Holt and McMillan [Bioinformatics 2014, ACM-BCB 2014]

last	Nodes	W		Nodes
0	$ $ $ $	G		$ $ $ $
1	.	T		.
1	? ? C A	A		? T C A
0	? ? C A	C		? ? C A
1	.	T		.
1	? G A	C		? G A
1	? ? T A	C•		? C A C
0	? ? A C	A	→	? T C C
1	.	T•		.
0	? ? A C	A		? ? A C
1	.	T•		.
0	? ? T C	A		? ? T C
1	.	C		.
1	? ? $ G	A		? ? $ G
1	? ? C G	#		? ? C G
1	? ? $ T	A		? ? $ T
1	? ? A T	C		? C A T

H&M Merging algorithm

- The H&M algorithm merges BWTs by progressively larger contexts.
- Very nice feature: only $2n$ bits working space (for Z^{h-1} and Z^h).

BWT	suffixes
b	$
c	ab$
$	abcab$
a	b$
a	bcab$
b	cab$

S_1 = abcab$

+

BWT	suffixes
c	#
#	aabcabc
c	abc#
a	abcabc#
a	bc#
a	bcabc#
b	c#
b	cabc#

S_2 = aabcabc#

=

BWT	suffixes
b	$
c	#
#	aabcabc
c	ab$
c	abc#
$	abcab$
a	abcabc#
a	b$
a	bc#
a	bcab$
a	bcabc#
b	c#
b	cab$
b	cabc#

H&M Merging algorithm

- The H&M algorithm merges BWTs by progressively larger contexts.
- Very nice feature: only 2n bits working space (for Z^{h-1} and Z^h).

Z	BWT	
0	b	$
0	c	ab$
0	$	abcab$
0	a	b$
0	a	bcab$
0	b	cab$
1	c	#
1	#	aabcabc
1	c	aabc#
1	a	abcabc#
1	a	bc#
1	a	bcabc#
1	b	c#
1	b	cabc#

h=1

Z	BWT	
0	b	$
1	c	#
0	c	ab$
0	$	abcab$
1	#	aabcabc
1	c	abc#
1	a	abcabc#
0	a	b$
0	a	bcab$
1	a	bc#
1	a	bcabc#
0	b	cab$
1	b	c#
1	b	cabc#

h=2

Z	BWT	
0	b	$
1	c	#
1	#	aabcabc
0	c	ab$
0	$	abcab$
1	c	abc#
1	a	abcabc#
0	a	b$
0	a	bcab$
1	a	bc#
1	a	bcabc#
1	b	c#
0	b	cab$
1	b	cabc#

h=3

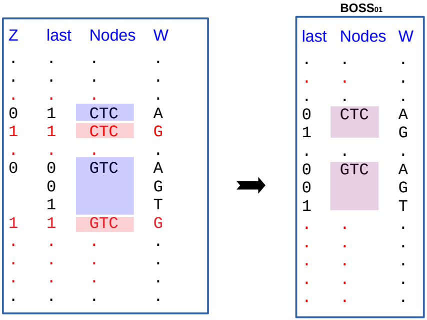
Z	BWT	
0	b	$
1	c	#
1	#	aabcabc
0	c	ab$
0	$	abcab$
1	c	abc#
1	a	abcabc#
0	a	b$
0	a	bcab$
1	a	bc#
1	a	bcabc#
1	b	c#
0	b	cab$
1	b	cabc#

dBG vs BWT Merging

- For dBGs we know the merging takes exactly k iterations (good)
- After the merging duplicate nodes have to be deleted (bad, extra work)

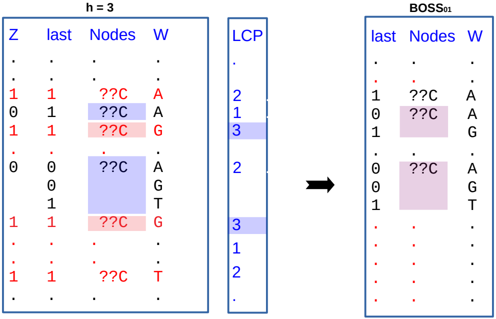
dBG vs BWT Merging

- For dBGs we know the merging takes exactly k iterations (good)
- After the merging duplicate nodes have to be deleted (bad, extra work)



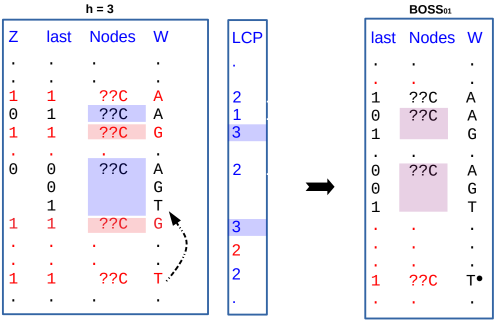
dBG vs BWT Merging

We can find duplicate nodes using right-to-left LCP values,



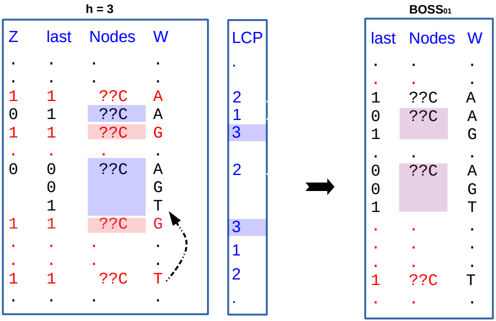
dBG vs BWT Merging

We can find duplicate nodes using right-to-left LCP values, that are also useful for finding which symbols in W should be marked with $\bullet$



dBG vs BWT Merging

We can find duplicate nodes using right-to-left LCP values, that are also useful for finding which symbols in W should be marked with •



Summing up

- With a modified H&M algorithm we co-lexicographically merge the nodes using only $2|V|$ bits of working space
- Using a trick from Egidi *et al.* [Spire '17], we compute the LCP values to detect duplicate nodes and correctly mark symbols in W
  - We can maintain only LCP values modulo 4 and only use additional $2|V|$ bits of working space
  - If we store the exact LCP values we get the Variable Order de Bruijn graph

Conclusions

- Algorithm for merging two of more BOSS succinct representations of de Bruijn graphs
- Cost: $\mathcal{O}(|E| + |V| \cdot k)$ time, where E and V are edges and vertices of the new graph, as in Muggli *et al.*
- Working space: $4|V|$ bits $+ \mathcal{O}(\sigma)$ words, less than Muggli *et al.*
- In external memory we can arrange the working space into 2σ distinct files so that all data is accessed sequentially
- We can support Variable Order and Colored variants.