

Dynamic quasi-minimal perfect hash function for k-mers



Work in Progress



Paola Bonizzoni¹, **Luca Denti**¹, Erik Garrison²,
Yuri Pirola¹ and Marco Previtali¹

¹Università degli studi di Milano - Bicocca

²University of California, Santa Cruz

February 5, 2020

Background and Motivation

A MPHf is an injective function $h : \{s_1, \dots, s_n\} \rightarrow [1, n]$

Static

Applications:

- Meraculous [Chapman et al. *PLoS one* (2011)]
- BCALM2 [Chikhi et al. *Bioinformatics* (2016)]
- pufferfish [Almodaresi et al. *Bioinformatics* (2018)]

Implementations:

- BBHash [Limasset et al. *SEA* (2017)]
- EMPHF [Belazzougui et al. *DCC* (2014)]

Dynamic

Applications:

- $\forall g$ [Garrison et al. *Nature* (2018)]

Implementations:

- ?

Background and Motivation

A MPHf is an injective function $h : \{s_1, \dots, s_n\} \rightarrow [1, n]$

Static

Applications:

- Meraculous [Chapman et al. PLoS one (2011)]
- BCALM2 [Chikhi et al. Bioinformatics (2016)]
- pufferfish [Almodaresi et al. Bioinformatics (2018)]

Implementations:

- BBHash [Limasset et al. SEA (2017)]
- EMPHF [Belazzougui et al. DCC (2014)]

Dynamic

Applications:

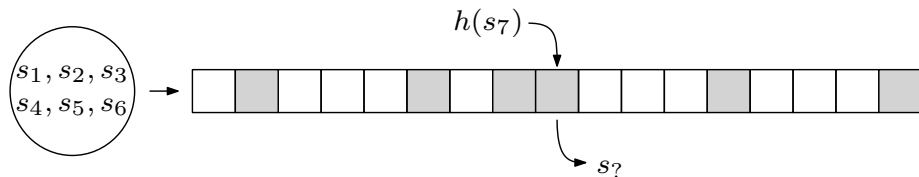
- vg [Garrison et al. Nature (2018)]

Implementations:

- ?

Challenge

How to retrieve an element when a collision occurs?

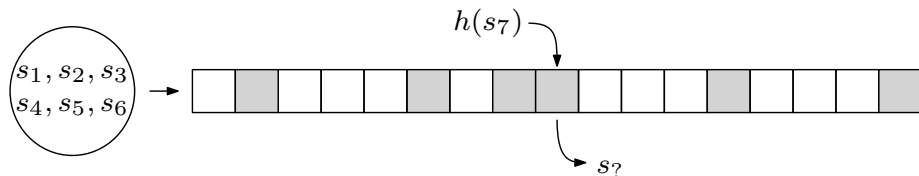


Our idea: combine a MPHF with a fully-dynamic de Bruijn graph

Our result: a dynamic MPHF for k -mers

Challenge

How to retrieve an element when a collision occurs?

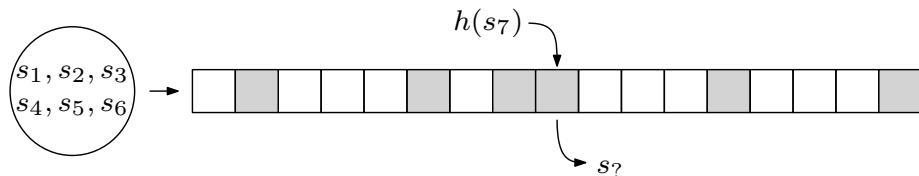


Our idea: combine a MPHF with a fully-dynamic de Bruijn graph

Our result: a dynamic MPHF for k -mers

Challenge

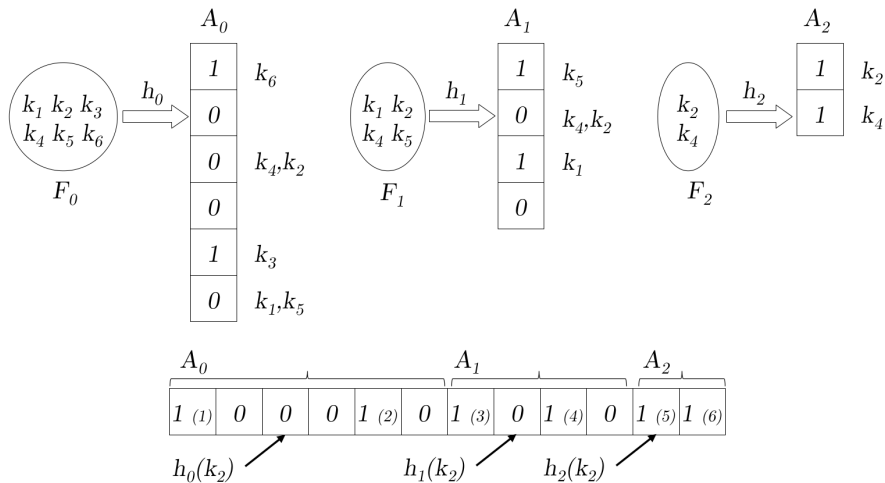
How to retrieve an element when a collision occurs?



Our idea: combine a MPHF with a fully-dynamic de Bruijn graph

Our result: a dynamic MPHF for k -mers

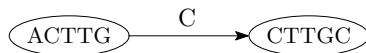
BBHash



[Image taken from Limasset et al. SEA (2017)]

Fully Dynamic de Bruijn graph

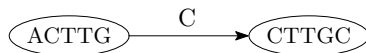
- bit matrices IN and OUT
- forest \mathcal{F} of spanning trees
- list R of roots



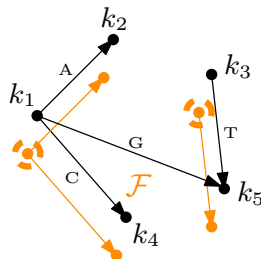
[Belazzougui et al. SPIRE (2016); Crawford et al. Bioinformatics (2018)]

Fully Dynamic de Bruijn graph

- bit matrices IN and OUT
- forest \mathcal{F} of spanning trees
- list R of roots



	IN				OUT			
	A	C	G	T	A	C	G	T
1 (k_3)	0	0	0	0	0	0	0	1
2 (k_1)	0	0	0	0	1	1	1	0
3 (k_4)	0	1	0	0	0	0	0	0
4 (k_2)	1	0	0	0	0	0	0	0
5 (k_5)	0	0	1	1	0	0	0	0



$$R = \langle k_1, k_3 \rangle$$

[Belazzougui et al. SPIRE (2016); Crawford et al. Bioinformatics (2018)]

Our idea

- the MPHF is a list of bit vectors (*blocks*)
- whenever we insert an element, we update the DBG
- the new elements are always added to the last block
- a new block is created if:
 - a) the last block is too “full”
 - b) a collision occurs in the last block
- if a collision occurs:
 1. recreate the element by visiting the DBG
 2. insert both elements

Our idea

- the MPHF is a list of bit vectors (*blocks*)
- whenever we insert an element, we update the dBG
- the new elements are always added to the last block
- a new block is created if:
 - a) the last block is too “full”
 - b) a collision occurs in the last block
- if a collision occurs:
 1. recreate the element by visiting the dBG
 2. insert both elements

Our idea

- the MPHf is a list of bit vectors (*blocks*)
- whenever we insert an element, we update the dBG
- the new elements are always added to the last block
- a new block is created if:
 - a) the last block is too “full”
 - b) a collision occurs in the last block
- if a collision occurs:
 1. recreate the element by visiting the dBG
 2. insert both elements

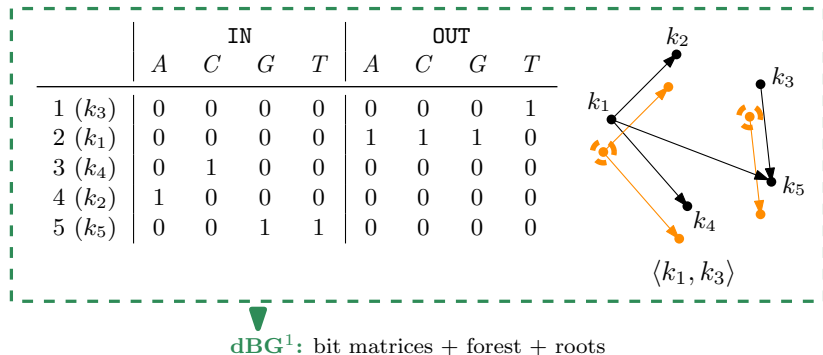
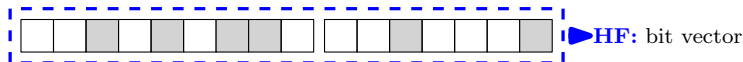
Our idea

- the MPHf is a list of bit vectors (*blocks*)
- whenever we insert an element, we update the dBG
- the new elements are always added to the last block
- a new block is created if:
 - a) the last block is too “full”
 - b) a collision occurs in the last block
- if a collision occurs:
 1. recreate the element by visiting the dBG
 2. insert both elements

Our idea

- the MPHF is a list of bit vectors (*blocks*)
- whenever we insert an element, we update the DBG
- the new elements are always added to the last block
- a new block is created if:
 - a) the last block is too “full”
 - b) a collision occurs in the last block
- if a collision occurs:
 1. recreate the element by visiting the DBG
 2. insert both elements

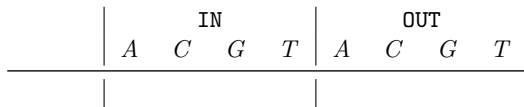
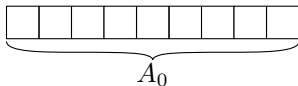
Structure overview



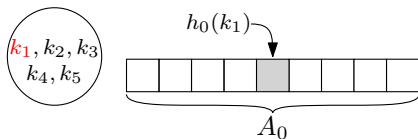
¹ Belazzougui et al. SPIRE (2016)

Insertion


k_1, k_2, k_3
 k_4, k_5



Insertion

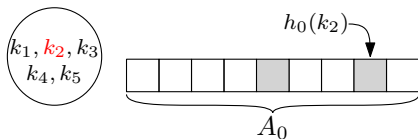


		IN				OUT				
		A	C	G	T	A	C	G	T	
▶	1 (k_1)	0	0	0	0	0	0	0	0	k_1

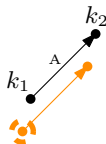


$\langle k_1 \rangle$

Insertion

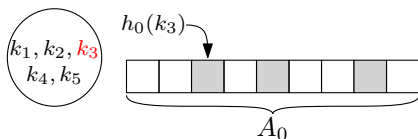


		IN				OUT			
		A	C	G	T	A	C	G	T
▶	1 (k_1)	0	0	0	0	1	0	0	0
	2 (k_2)	1	0	0	0	0	0	0	0

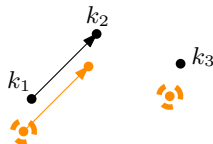


$\langle k_1 \rangle$

Insertion

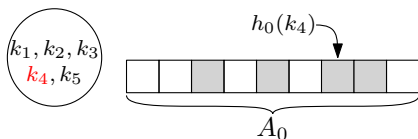


		IN				OUT			
		A	C	G	T	A	C	G	T
▶	1 (k_3)	0	0	0	0	0	0	0	0
	2 (k_1)	0	0	0	0	1	0	0	0
	3 (k_2)	1	0	0	0	0	0	0	0

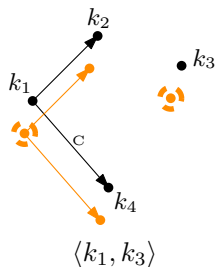


$$\langle k_1, k_3 \rangle$$

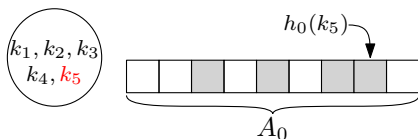
Insertion



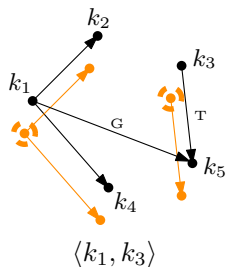
	IN				OUT			
	A	C	G	T	A	C	G	T
1 (k_3)	0	0	0	0	0	0	0	0
2 (k_1)	0	0	0	0	1	1	0	0
3 (k_4)	0	1	0	0	0	0	0	0
4 (k_2)	1	0	0	0	0	0	0	0



Insertion

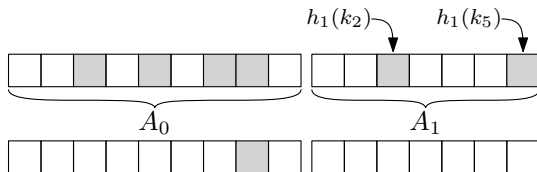


	IN				OUT			
	A	C	G	T	A	C	G	T
1 (k_3)	0	0	0	0	0	0	0	0
2 (k_1)	0	0	0	0	1	1	0	0
3 (k_4)	0	1	0	0	0	0	0	0
4 (k_2)	1	0	0	0	0	0	0	0

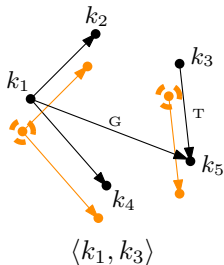


Insertion

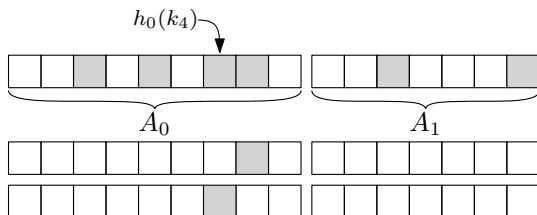
k_1, k_2, k_3
 k_4, k_5



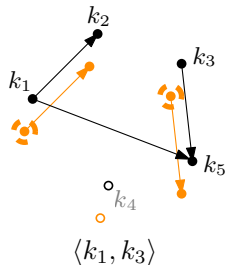
		IN				OUT			
		A	C	G	T	A	C	G	T
1	(k_3)	0	0	0	0	0	0	0	1
2	(k_1)	0	0	0	0	1	1	1	0
3	(k_4)	0	1	0	0	0	0	0	0
4	(k_2)	1	0	0	0	0	0	0	0
▶	5 (k_2)	1	0	0	0	0	0	0	0
▶	6 (k_5)	0	0	1	1	0	0	0	0



Deletion



	IN				OUT			
	A	C	G	T	A	C	G	T
1 (k_3)	0	0	0	0	0	0	0	1
2 (k_1)	0	0	0	0	1	0	1	0
3 (k_4)	0	1	0	0	0	0	0	0
4 (k_2)	1	0	0	0	0	0	0	0
5 (k_2)	1	0	0	0	0	0	0	0
6 (k_5)	0	0	1	1	0	0	0	0



Conclusions and Open Questions

Dynamic implementation of

- quasi-minimal perfect hash function for k -mers
- de Bruijn graph

OQs:

- is it reasonable and efficient in practice?
- can we also find good theoretical bounds?
- how many shifts should we allow?
- when should we create a new block?
- worst case: 2^{b-1} insertions in the last block (but in practice?) *[b blocks]*
- when should we rebuild everything? can we reuse some parts?

Conclusions and Open Questions

Dynamic implementation of

- quasi-minimal perfect hash function for k -mers
- de Bruijn graph

OQs:

- is it reasonable and efficient **in practice?**
- can we also find good theoretical bounds?
- how many shifts should we allow?
- when should we create a new block?
- worst case: 2^{b-1} insertions in the last block (but in practice?) *[b blocks]*
- when should we rebuild everything? can we reuse some parts?

Conclusions and Open Questions

Dynamic implementation of

- quasi-minimal perfect hash function for k -mers
- de Bruijn graph

OQs:

- is it reasonable and efficient **in practice**?
- can we also find good theoretical bounds?
- how many shifts should we allow?
- when should we create a new block?
- worst case: 2^{b-1} insertions in the last block (but in practice?) *[b blocks]*
- when should we rebuild everything? can we reuse some parts?

Conclusions and Open Questions

Dynamic implementation of

- quasi-minimal perfect hash function for k -mers
- de Bruijn graph

OQs:

- is it reasonable and efficient **in practice?**
- can we also find good theoretical bounds?
- how many shifts should we allow?
- when should we create a new block?
- worst case: 2^{b-1} insertions in the last block (but in practice?) *[b blocks]*
- when should we rebuild everything? can we reuse some parts?

Conclusions and Open Questions

Dynamic implementation of

- quasi-minimal perfect hash function for k -mers
- de Bruijn graph

OQs:

- is it reasonable and efficient **in practice?**
- can we also find good theoretical bounds?
- how many shifts should we allow?
- when should we create a new block?
- worst case: 2^{b-1} insertions in the last block (but in practice?) *[b blocks]*
- when should we rebuild everything? can we reuse some parts?

Conclusions and Open Questions

Dynamic implementation of

- quasi-minimal perfect hash function for k -mers
- de Bruijn graph

OQs:

- is it reasonable and efficient *in practice*?
- can we also find good theoretical bounds?
- how many shifts should we allow?
- when should we create a new block?
- worst case: 2^{b-1} insertions in the last block (but in practice?) *[b blocks]*
- when should we rebuild everything? can we reuse some parts?

Thank You!
Questions? Suggestions? Idea?
Wanna help us?