Cuttlefish: Fast, parallel, and low-memory compaction of de Bruijn graphs from large-scale genome collections

DSB'2021

Jamshed Khan and Rob Patro

University of Maryland, College Park, MD

Thu 11th Feb, 2021

Context

With the increasing throughput of sequencing, we now have—

- thousands of mammalian genomes
- genomes orders of magnitude larger to typical mammalian genomes, e.g. the sugar pine (~31 Gbp) and the mexican walking fish (~32 Gbp)

Context

With the increasing throughput of sequencing, we now have—

- thousands of mammalian genomes
- genomes orders of magnitude larger to typical mammalian genomes, e.g. the sugar pine (~31 Gbp) and the mexican walking fish (~32 Gbp)

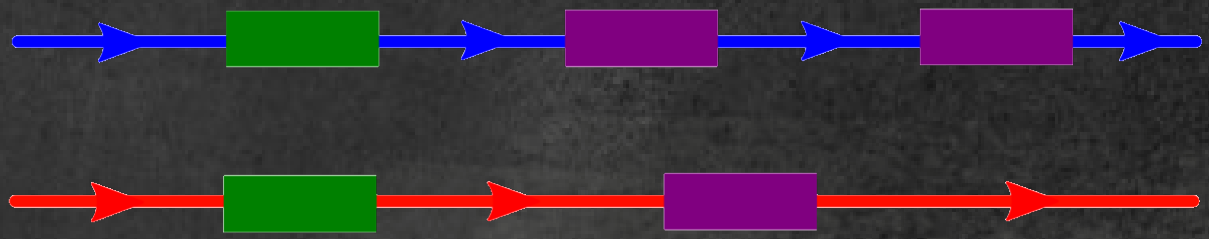
There is an increasing need for (efficient) —

- indices
- representations for comparative genomics and pan-genome analysis

Context

Reference representations

Simplest form—

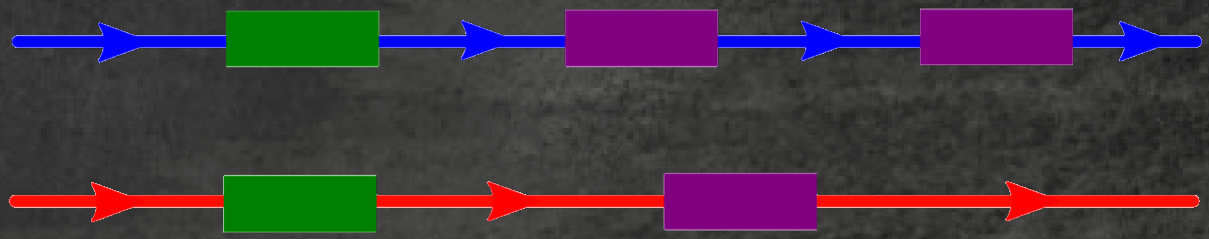


Images are borrowed from Ilia Minkin.

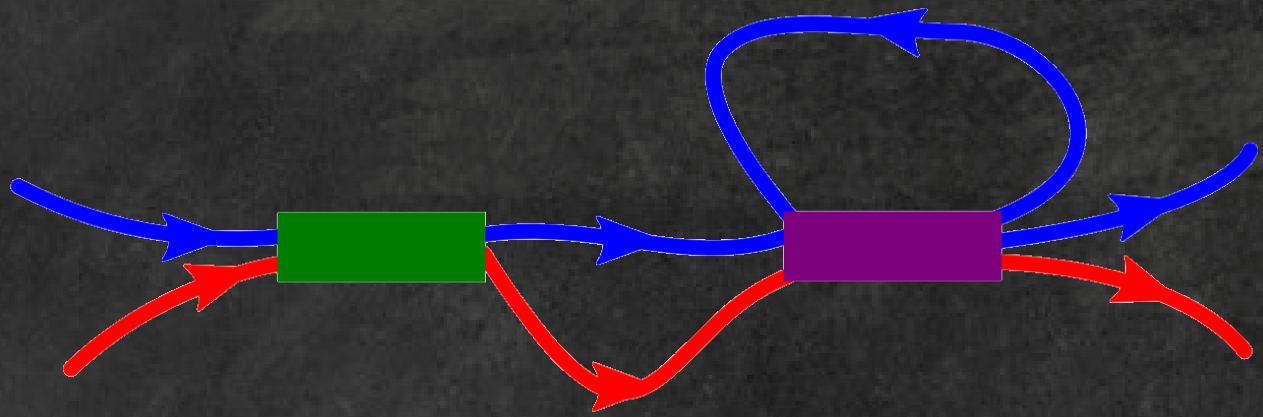
Context

Reference representations

Simplest form—



What we (may) intend to have—



Images are borrowed from Ilia Minkin.

Context

De Bruijn Graphs

An excellent object to represent genome references (and sequencing reads).

Context

De Bruijn Graphs

An excellent object to represent genome references (and sequencing reads).

And the (colored) compacted variant is of specific interest to us.

Context
De Bruijn Graphs

$k = 2$

TGACGTC

TGACTTC

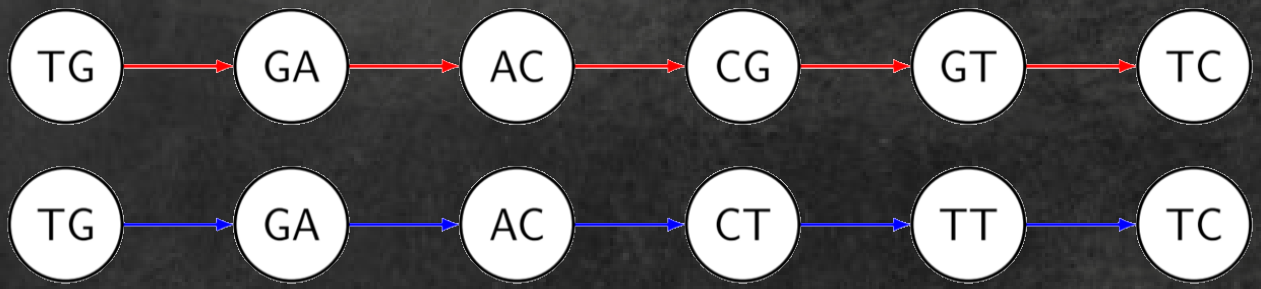
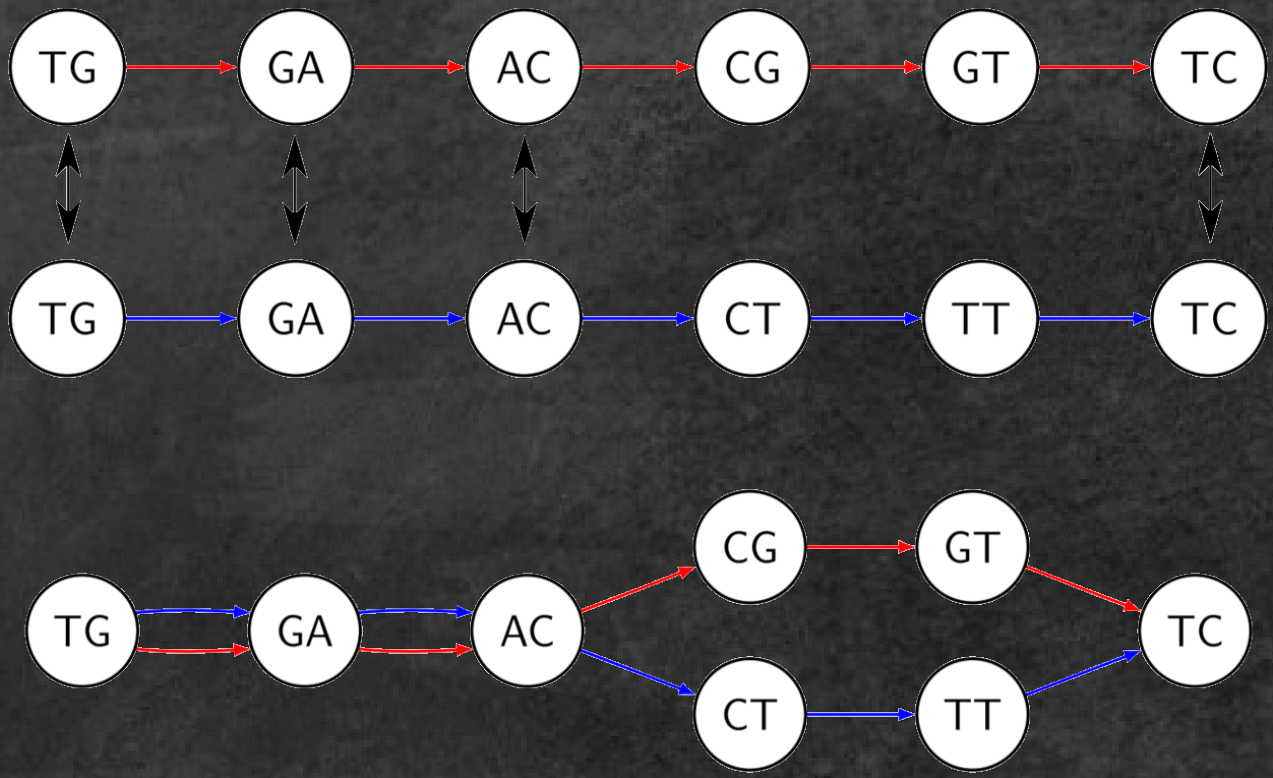


Image borrowed from Ilia Minkin.

Context
De Bruijn graphs



Images borrowed from Ilia Minkin.

Context

De Bruijn Graphs

We tackle the initial steps of whole-genome analysis pipelines—time- and memory-efficient construction of the (colored) compacted de Bruijn graphs.

Preliminaries
Bidirected de Bruijn graphs

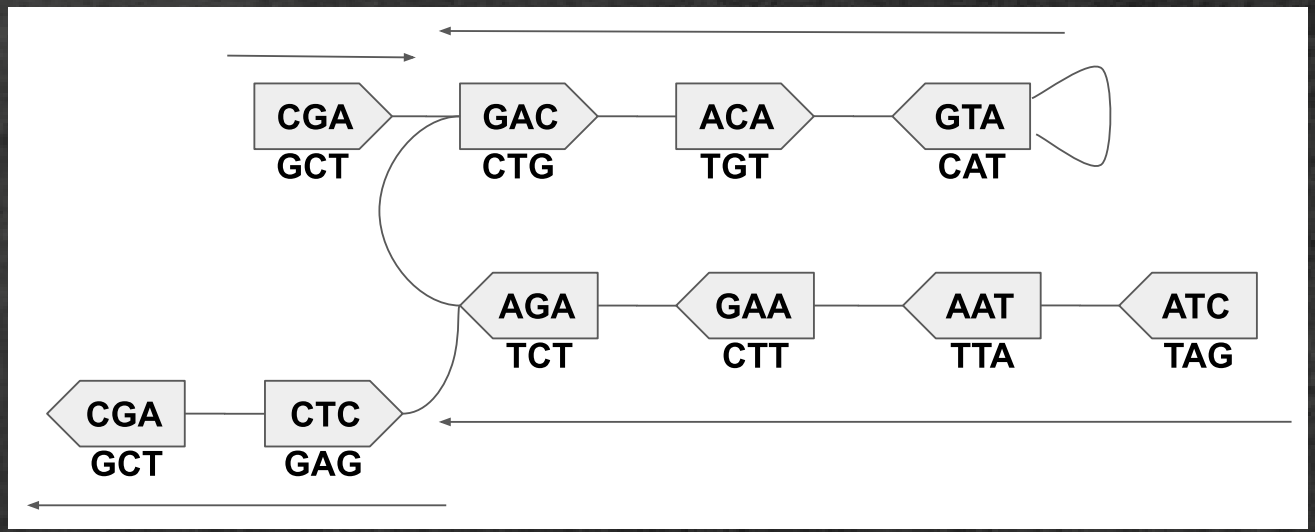


Figure: $G(S, k)$ for $S = \{CGACATGTCTTAG, GCTCTTAG\}$ with $k = 3$.

Each vertex (canonical k-mer) has two sides: front and back.

Preliminaries

Compacted bidirected de Bruijn graphs

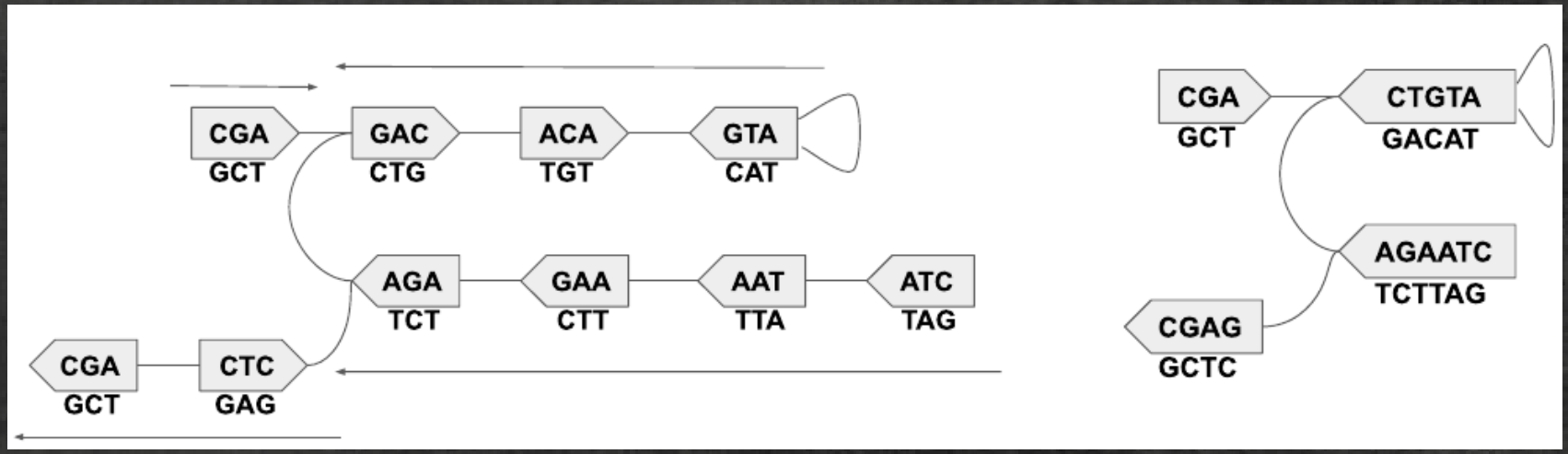


Figure: $G(S, k)$ and $G_c(S, k)$.

Algorithm
Motivation

Given a set of references R —

Naïve-Compaction(R)

1 G = Construct-de-Bruijn-Graph(R)
2 $\mathsf{G_c}$ = Compact-Using-Linear-Traversal(G)
3 return $\mathsf{G_c}$

Algorithm
Motivation

Given a set of references R —

Naïve-Compaction(R)

1 G = Construct-de-Bruijn-Graph(R)
2 $\mathsf{G_c}$ = Compact-Using-Linear-Traversal(G)
3 return $\mathsf{G_c}$

- Infeasible—the space requirements are enormous.
- Need to bypass the $\mathsf{G}(\mathsf{R}, \mathsf{k})$ construction.

Algorithm
Motivation

Implicit walk over $\mathsf{G}(\mathsf{S}, \mathsf{k})$:

Algorithm
Motivation

Implicit walk over $\mathsf{G}(\mathsf{S}, \mathsf{k})$:

- For an edge-centric de Bruijn graph, a complete walk traversal $w(s)$ over $\mathsf{G}(s, \mathsf{k})$ can be obtained through a scan over s, without having $\mathsf{G}(s, \mathsf{k})$;

Algorithm
Motivation

Implicit walk over $\mathsf{G}(\mathsf{S}, \mathsf{k})$:

- For an edge-centric de Bruijn graph, a complete walk traversal $w(s)$ over $\mathsf{G}(\mathsf{s}, \mathsf{k})$ can be obtained through a scan over s, without having $\mathsf{G}(\mathsf{s}, \mathsf{k})$;
- and the maximal unitigs of $\mathsf{G}(\mathsf{s}, \mathsf{k})$ are contained as subpaths in this walk.

Algorithm

Flanking vertices of the maximal unitigs

- If each vertex in $\mathsf{G}(\mathsf{S}, \mathsf{k})$ can be characterized as flanking or internal (w.r.t to the maximal unitigs), the unitigs themselves can then be extracted through identifying subpaths having flanking vertices at both ends.

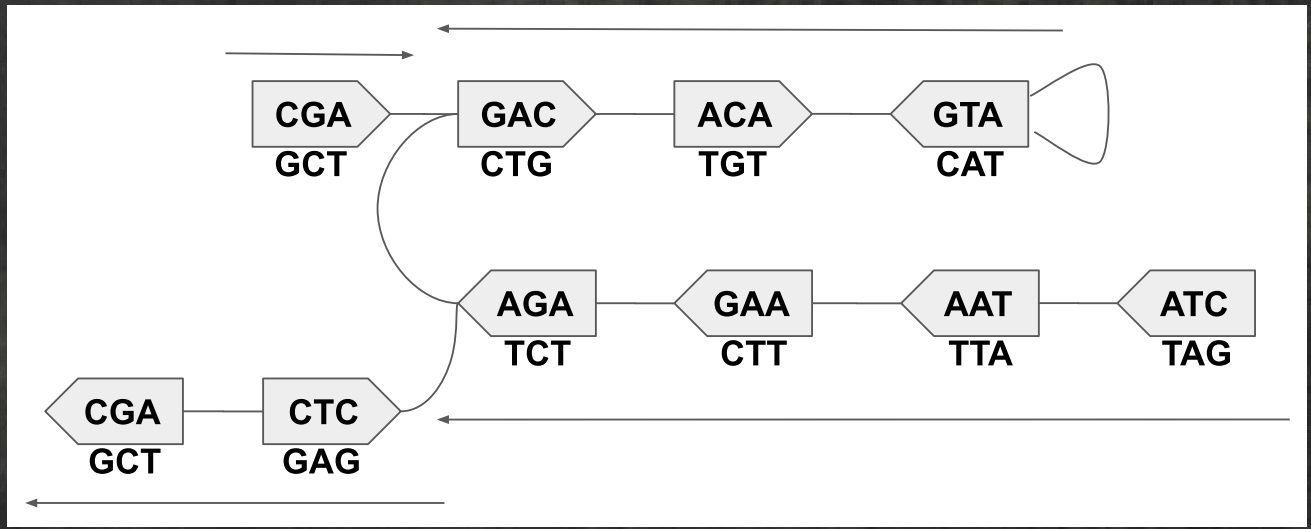


Figure: $\mathsf{CGA}, \mathsf{GAC}, \mathsf{ATG}, \mathsf{AGA}, \mathsf{CTA}, \mathsf{AGC},$ and CTC are flanking.

Algorithm

Flanking vertices of the maximal unitigs

- If each vertex in $\mathsf{G}(\mathsf{S}, \mathsf{k})$ can be characterized as flanking or internal (w.r.t to the maximal unitigs), the unitigs themselves can then be extracted through identifying subpaths having flanking vertices at both ends.

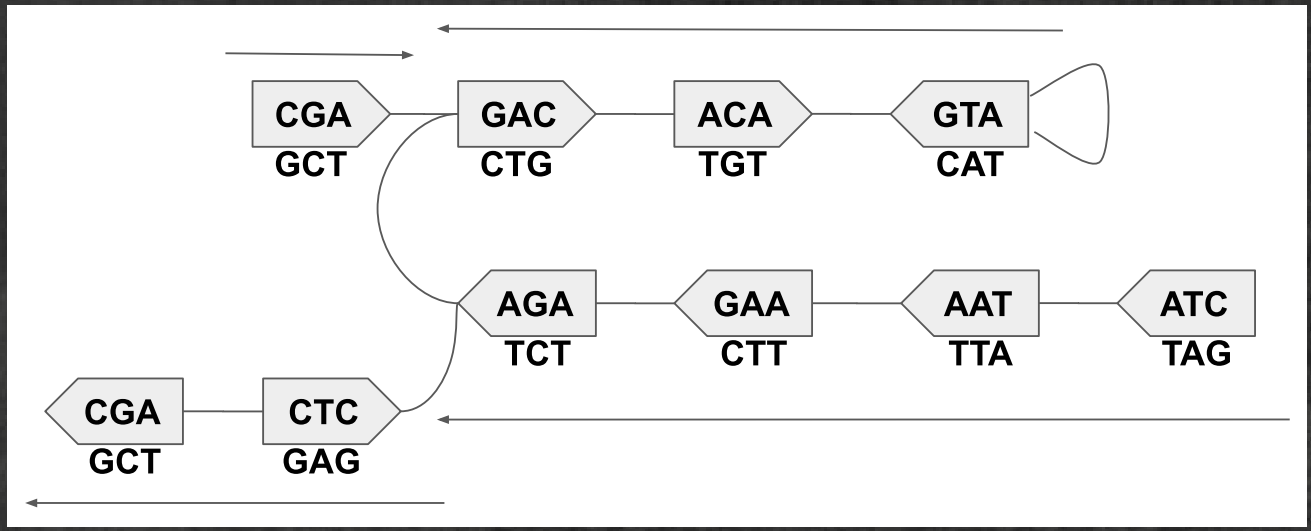


Figure: $\mathsf{CGA}, \mathsf{GAC}, \mathsf{ATG}, \mathsf{AGA}, \mathsf{CTA}, \mathsf{AGC},$ and CTC are flanking.

- Thus the graph compaction problem can be reduced to the problem of determining the set of the flanking vertices.

Algorithm

Flanking vertices of the maximal unitigs

A vertex v in $G(S, k)$ is referred to as a flanking vertex if it has a side s_v with —

1. 0 or >1 incident edges
2. or, exactly one edge (v, s_v, u, s_u), and s_u has >1 incident edges.

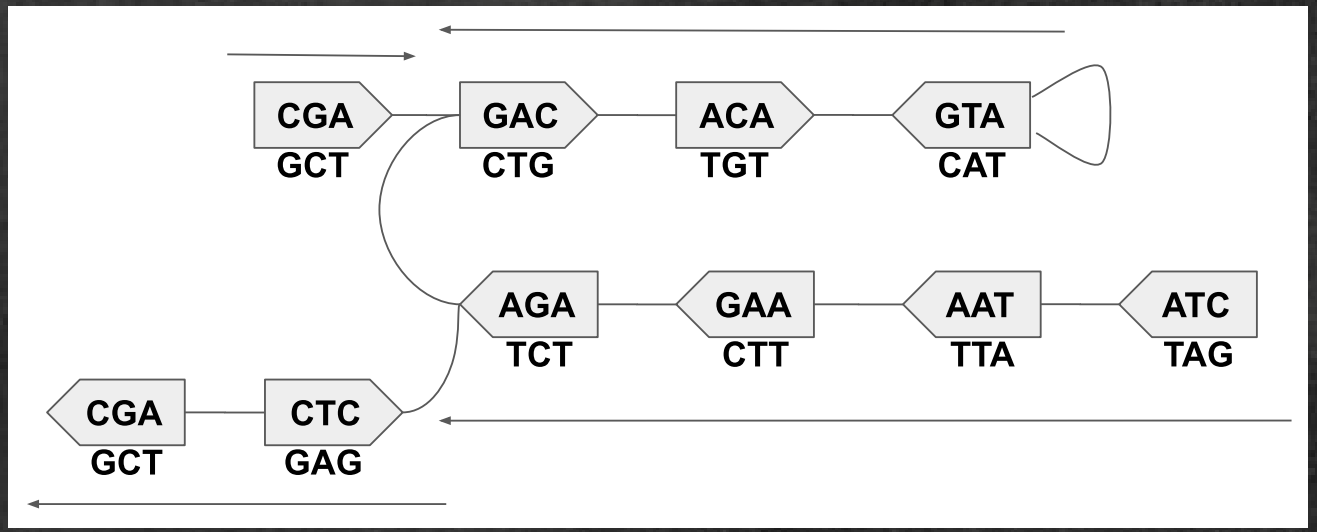


Figure: CGA, GAC, ATG, AGA, CTA, AGC, and CTC are flanking.

Algorithm

A DFA model for the vertices

In $\mathsf{G}(\mathsf{S}, \mathsf{k})$, a side of a vertex can be considered to be in five different configurations—

- one for each unique singleton edge;
- one for when it has $\neq 1$ distinct edges.

Any other adjacency information is irrelevant for our purposes.

Algorithm

A DFA model for the vertices

In $\mathsf{G}(\mathsf{S}, \mathsf{k})$, a side of a vertex can be considered to be in five different configurations—

- one for each unique singleton edge;
- one for when it has $\neq 1$ distinct edges.

Any other adjacency information is irrelevant for our purposes.

Thus, a vertex can be in $(5 \times 5) = 25$ configurations (or, states).

Algorithm

A DFA model for the vertices

Each vertex is treated as a deterministic finite-state automaton (DFA) $(Q, \Sigma', \delta, q_0, Q')$:

Algorithm

A DFA model for the vertices

Each vertex is treated as a deterministic finite-state automaton (DFA) $(Q, \Sigma', \delta, q_0, Q')$:

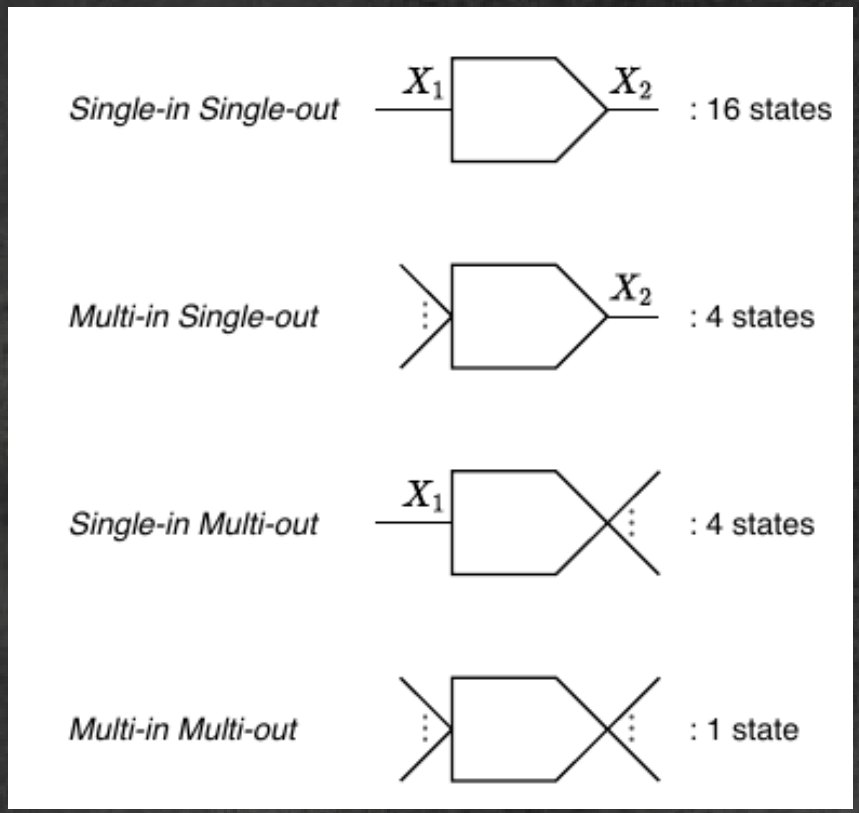
- q_0 is the initial state of the DFA—the unvisited state.

Algorithm

A DFA model for the vertices

Each vertex is treated as a deterministic finite-state automaton (DFA) $(Q, \Sigma', \delta, q_0, Q')$:

- Q' is the set of possible 25 states, which can be partitioned into four disjoint classes:



Algorithm

A DFA model for the vertices

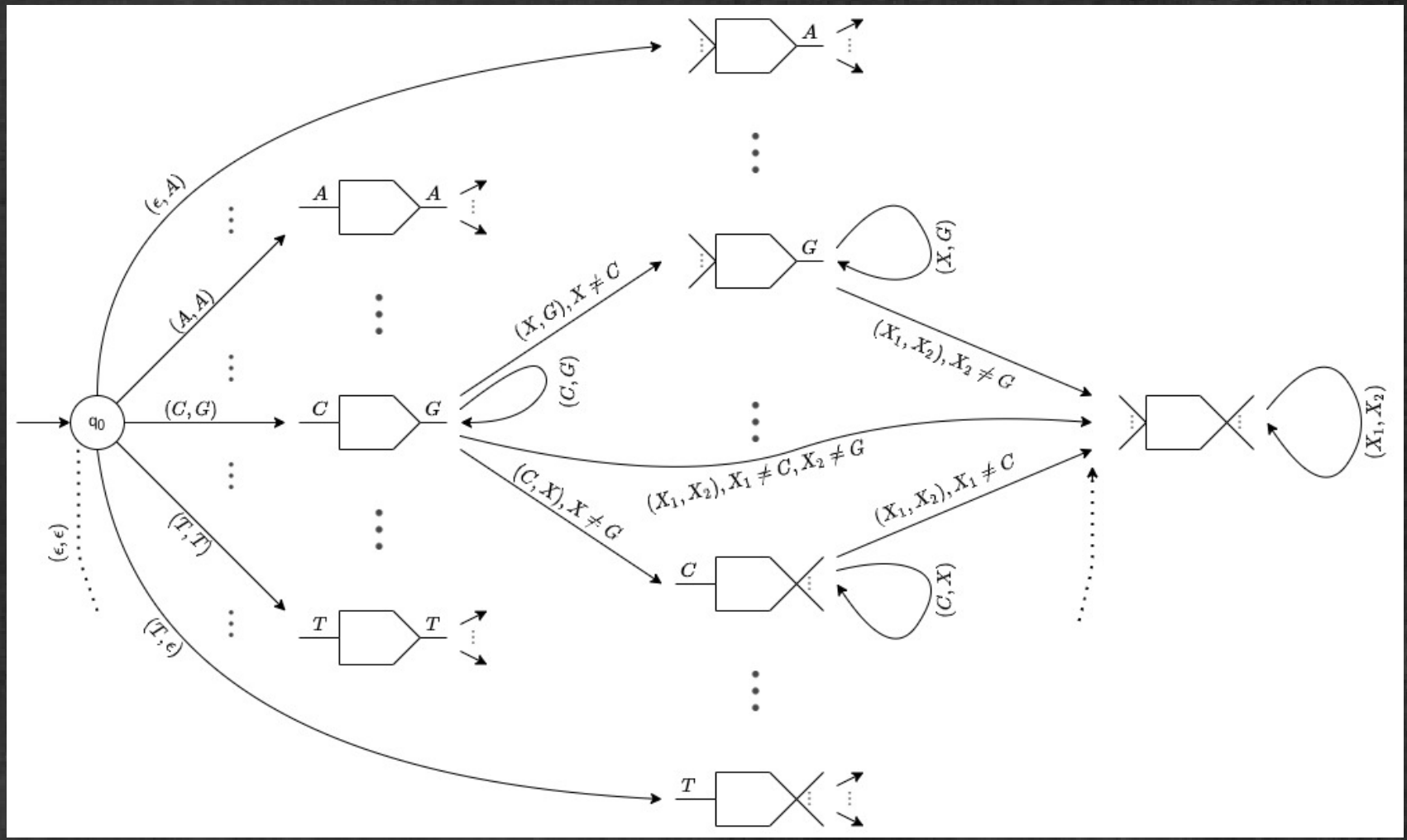
Each vertex is treated as a deterministic finite-state automaton (DFA) $(Q, \Sigma', \delta, q_0, Q')$:

- δ is the transition function.

Algorithm

A DFA model for the vertices

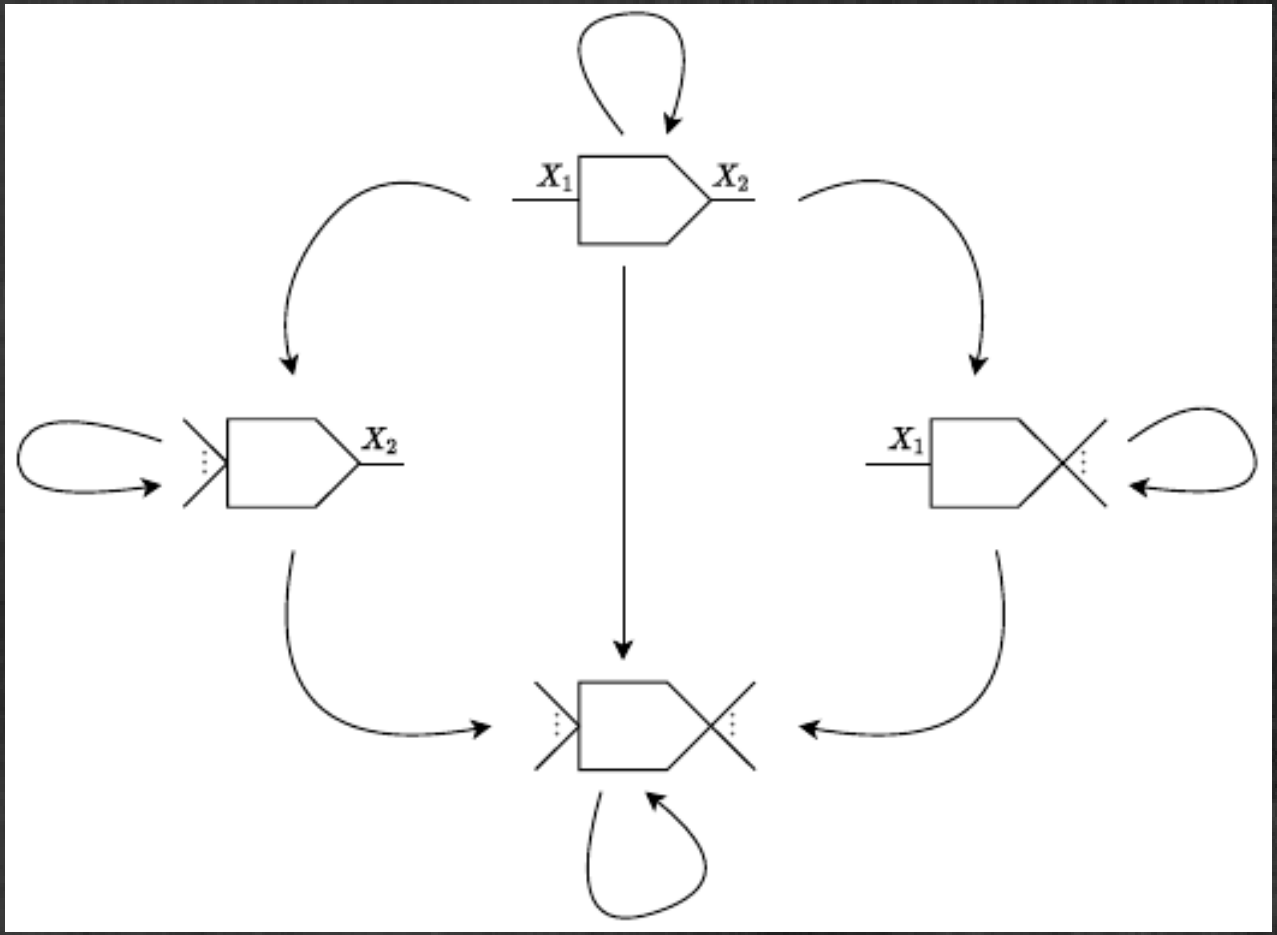
The transition function δ –



Algorithm

A DFA model for the vertices

A high-level view of the possible types of transitions between states of the four classes:



Algorithm

Hash table structure for the automata

We maintain—

1. a minimal perfect hash as the hash function(`BBHash` [4]) (the set K of keys are static) — taking ~3.7 bits/k-mer [5]
2. a bit-packed array for the hash buckets — taking $\lceil \log_2(26) \rceil = 5$ bits/k-mer

[4] Limasset, A. et al. (2017)

[5] in theory, this could be < 2 bits/k-mer using a more compact MPHF

Algorithm

The Cuttlefish algorithm

For a set S of input strings—

Cuttlefish(S)

```
1  K = Extract-Unique-k-mers(S)
2  h = Construct-Minimal-Perfect-Hash-Function(K)
3  B = Compute-States(S, h, |K|)
4  for each s ∈ S
5        Extract-Maximal-Unitigs(s, h, B)
```

Algorithm
Asymptotics

For a reference collection R with total length m and n distinct k-mers:

- the running time is $O((m+n)(\lceil k/32 \rceil + h))$, h being an expected constant
- the memory usage is $\Theta(8.7n) = \Theta(n)$

Results

Dataset characteristics

Individual genome references from —

- human (~ 3.2 Gbp);
- western gorilla (~ 3 Gbp);
- and sugar pine (~ 27.6 Gbp).

Collections of references —

- 62 E. Coli (~ 310 Mbp);
- 7 humans (~ 21 Gbp);
- 7 apes (~ 18 Gbp);
- 11 conifer plants (~ 204 Gbp);
- 100 humans (~ 322 Gbp).

Results

Benchmarking comparisons

Dataset	Thread-count	k	Bifrost	deGSM	TwoPaCo	Cuttlefish
			Build	Build	Build	Build
Human	1	31	04:54:50 (27.23)	01:54:41 (37.94)	01:13:19 (4.15)	**32:59** (**2.79**)
		61	05:16:51 (50.19)	02:20:57 (84.16)	01:10:18 (6.02)	**38:21** (**3.06**)
	8	31	01:33:54 (27.23)	25:20 (37.94)	12:57 (5.04)	**05:49** (**2.79**)
		61	01:20:28 (50.18)	47:52 (84.16)	11:28 (5.46)	**07:45** (**3.06**)
	16	31	01:24:40 (27.24)	18:19 (37.94)	06:24 (5.57)	**03:26** (**2.79**)
		61	01:12:33 (50.18)	46:34 (84.16)	07:12 (5.55)	**04:23** (**3.06**)
Gorilla	1	31	05:44:10 (28.08)	01:34:29 (37.94)	01:00:15 (5.04)	**31:46** (**2.74**)
		61	05:31:06 (50.13)	02:11:33 (84.16)	01:11:29 (5.83)	**38:15** (**3.02**)
	8	31	02:06:52 (28.08)	28:52 (37.94)	13:02 (5.82)	**05:30** (**2.74**)
		61	01:24:21 (50.13)	47:45 (84.16)	10:03 (6.00)	**07:58** (**3.02**)
	16	31	01:50:26 (28.08)	20:47 (37.94)	07:29 (5.52)	**03:13** (**2.74**)
		61	01:10:06 (50.13)	38:45 (84.16)	06:24 (6.09)	**04:29** (**3.02**)
Sugar pine	16	31	22:18:24 (229.17)	09:29:24 (145.23)	01:49:01 (61.93)	**51:30** (**14.24**)
		61	X (364.25)	X (166.54)	**01:26:39** (64.86)	03:14:44 (**20.88**)

Time- and memory-performance benchmarking for compacting single input reference de Bruijn graphs. Running times are in wall clock format, and the maximum memory usages in gigabytes. Note that, **Bifrost** and **deGSM** can also work with sequencing reads.

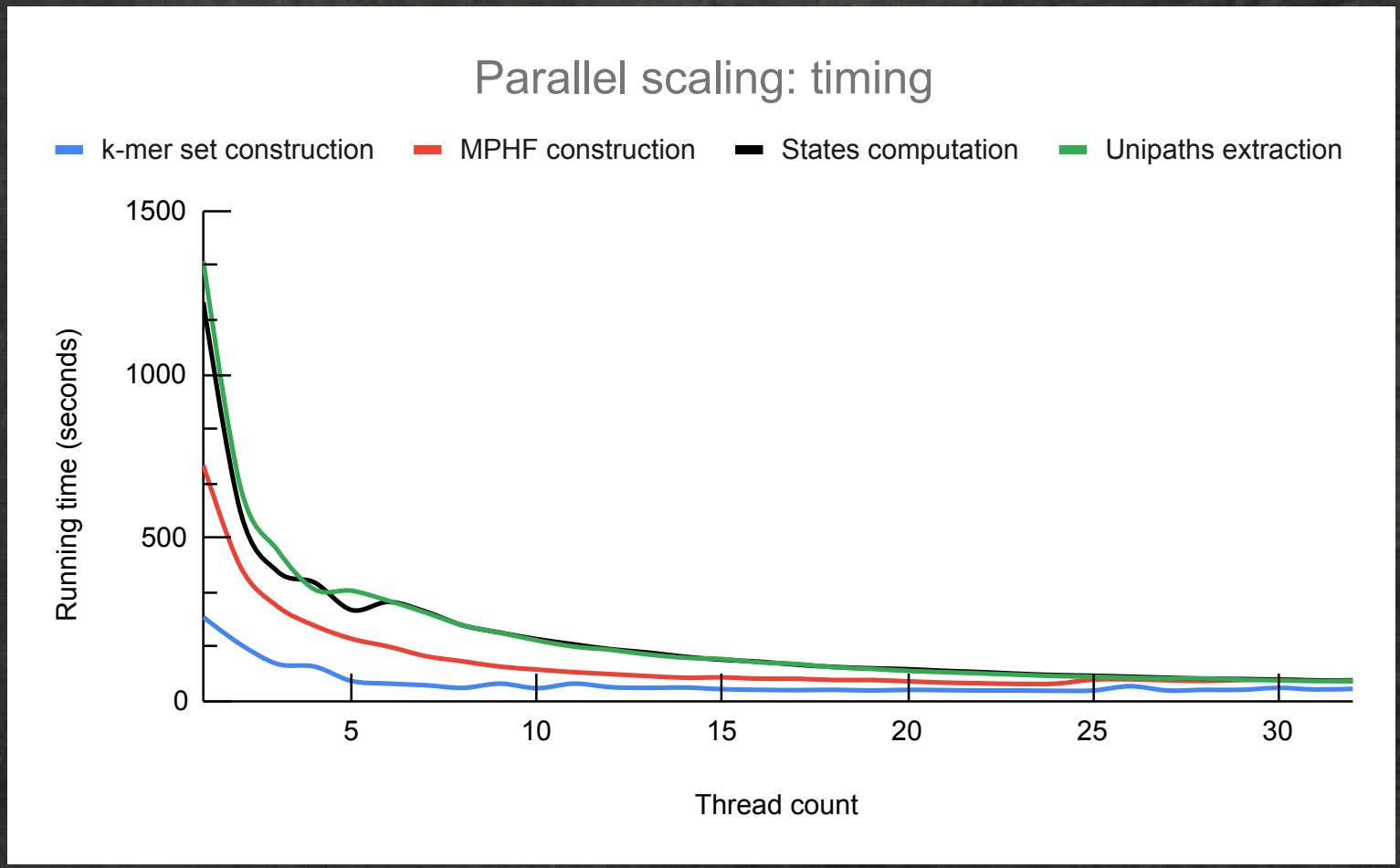
Results

Benchmarking comparisons

Dataset	Total genome-length (bp)	Distinct kmers count	Bifrost	deGSM	TwoPaCo	Cuttlefish
62 E. Coli	310M	24M	1 (**0.47**)	1 (3.34)	1 (0.80)	1 (0.96)
7 Humans	21G	2.6B	95 (29.06)	30 (37.94)	62 (6.14)	**21** (**2.88**)
7 Apes	18G	7.1B	294 (100.25)	172 (145.23)	59 (28.87)	**25** (**7.42**)
11 Conifers	204G	82B	–	–	981 (288.99)	**525** (**84.12**)
100 Humans	322G	28B	–	–	X (64.88)	**523** (**28.75**)

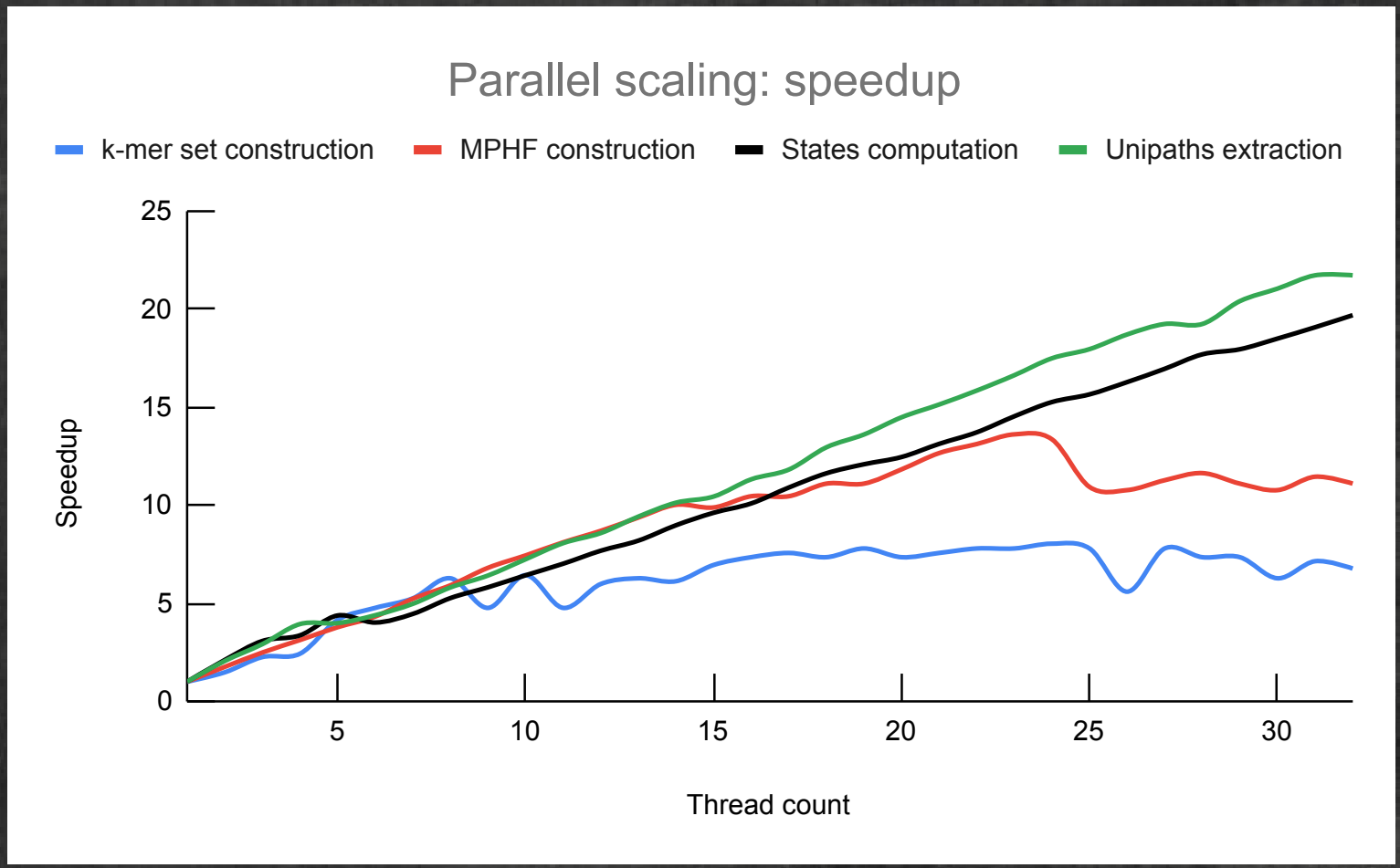
Time- and memory-performance benchmarking for compacting colored de Bruijn graphs (i.e. multiple input references) for $k = 31$, using 16 threads. Running times are minutes, and the maximum memory usages in gigabytes.

Results
Parallel scalability



Time taken by each step.

Results
Parallel scalability



Speedup for each step.

Conclusion

- Pushing the boundary of the ability to construct (colored) compacted dBGs, in terms of genome scale and count.
- Introduction of a novel modeling scheme of the dBG vertices with a DFA.
- Potential further improvements in the role of the dBG in—
  - comparative genomics, computational pan-genomics, and sequence analysis pipelines;
  - also facilitating novel biological studies — especially for large-scale genome collections that may not have been possible earlier.
- Implemented using `C++14`, available at https://github.com/COMBINE-lab/cuttlefish.

Appendix I

A key observation to bypass building $\mathsf{G}(\mathsf{S}, \mathsf{k})$ —

- A complete walk traversal $w(\mathsf{s})$ over $\mathsf{G}(\mathsf{s}, \mathsf{k})$ can be obtained through a scan over s, without having $\mathsf{G}(\mathsf{s}, \mathsf{k})$.

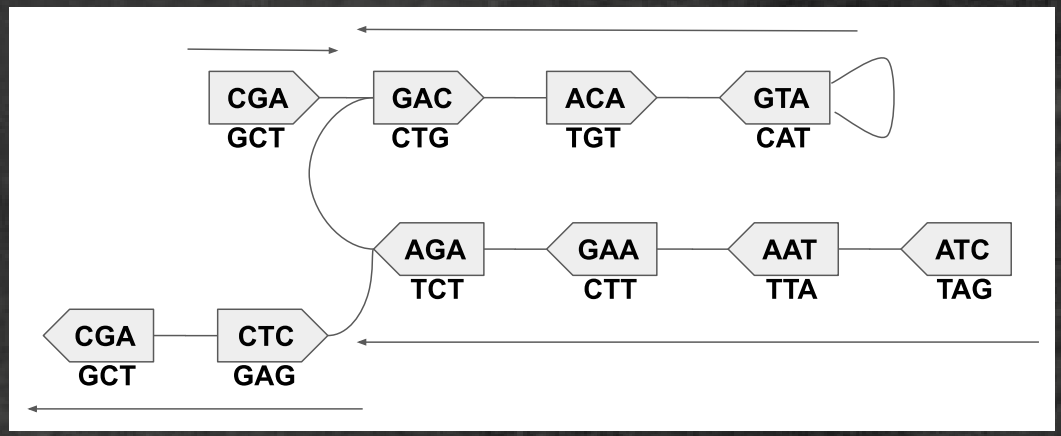


Figure: $\mathsf{G}(\mathsf{S}, 3)$ for $\mathsf{S} = \{\mathsf{CGACATGTCTTAG}, \mathsf{GCTCTTAG}\}$.

The string $\mathsf{GCTCTTAG}$ is spelled by the walk $(\mathsf{AGC}, \mathsf{CTC}, \mathsf{AGA}, \mathsf{GAA}, \mathsf{TAA}, \mathsf{CTA})$.

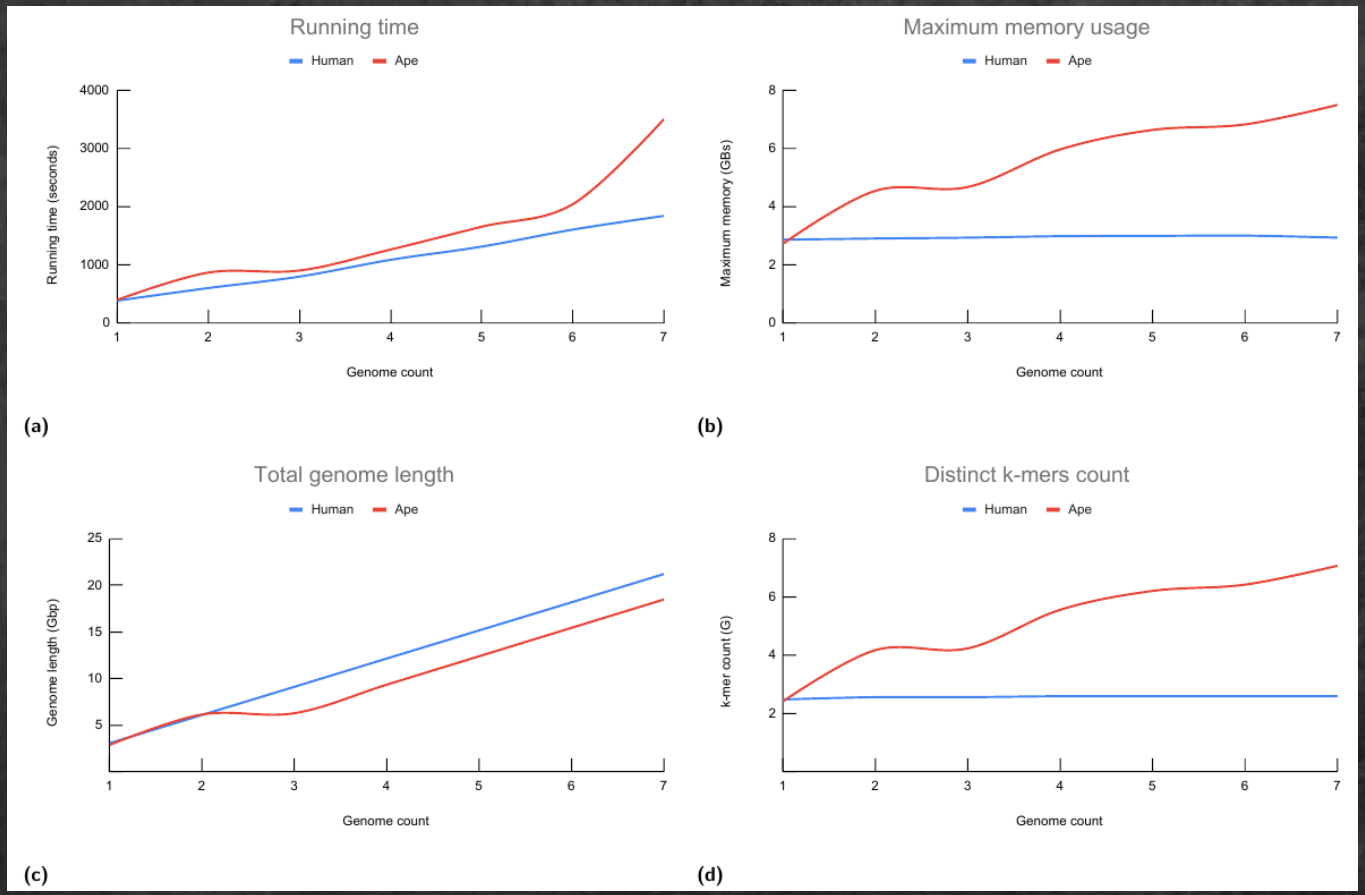
Appendix II

Scaling with k

k	Distinct k-mers count (B)	Build steps (s)			Build Time (s)	Build memory (GB)	Output step (s)		Output memory (GB)
		k-mer set construction	MPHF construction	States computation			Unipaths only	GFA2	
23	2.39	154	62	762	978	2.67	744	1345	2.82
31	2.59	391	70	791	1252	2.88	737	1203	3.01
61	2.96	439	200	797	1436	3.25	798	831	3.37
91	3.12	1118	311	830	2259	3.42	806	860	3.49
121	3.24	1483	902	841	3226	3.55	850	820	3.62

Running times are in seconds, and the maximum memory usages are in gigabytes.

Appendix III



For genome counts varying from 1 to 7, the corresponding (a) running time (seconds), (b) maximum memory usage (gigabytes), (c) total length of the genomes. and (d) number distinct k-mers for each input collection.