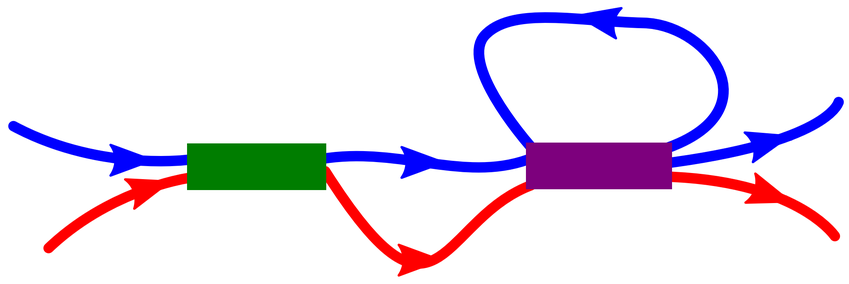
TwoPaCo: An efficient algorithm to build the compacted de Bruijn graph from many complete genomes

Ilia Minkin[1], Son Pham[2], Paul Medvedev[1,3,4]

[1]Department of Computer Science and Engineering, The Pennsylvania State University, USA; [2]Salk Institute for Biological Studies, USA;
[3]Department of Biochemistry and Molecular Biology, The Pennsylvania State University, USA;
[4]Genomic Sciences Institute of the Huck, The Pennsylvania State University, USA

Graph Representation of Genomes

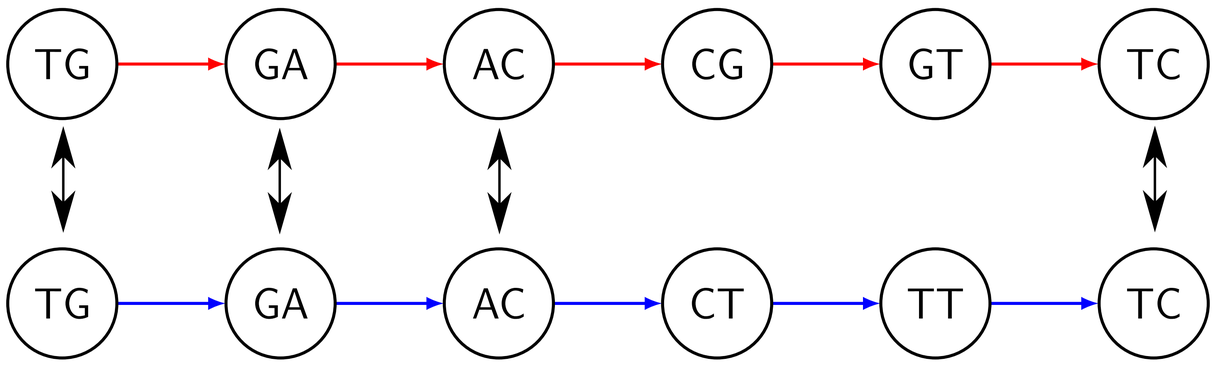
As we get more and more complete genomes, the question of their representation gets more important. Unlike a linear representation, graphs can capture homology both within and between genomes. This property makes them useful for comparative genomics applications. An example of graph representation of two genomes is shown below. Red and blue lines are sequences, rectangles are homologous blocks:



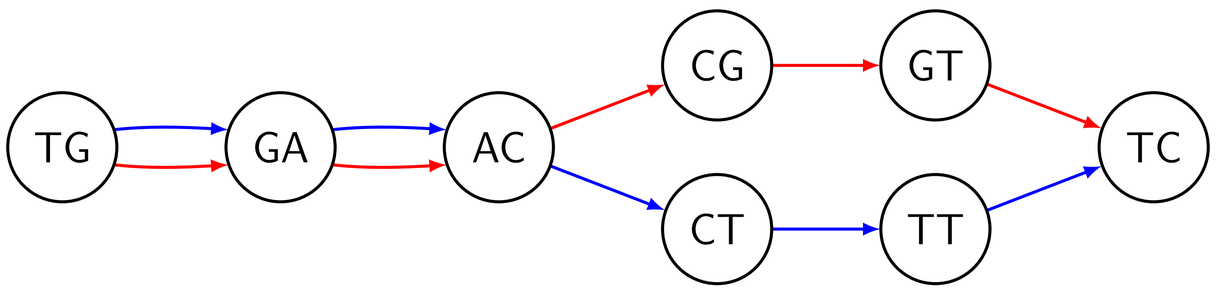
A possible graph model is the **compacted de Bruijn graph** that recently demonstrated utility in synteny identification. Unfortunately, construction of this graph is prohibitive for large inputs: the fastest algorithm to date was able to process seven whole mammalian genomes in under eight hours [2].

De Bruijn Graph Construction

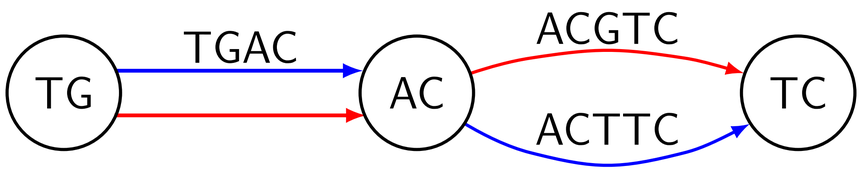
The de Bruijn graph is defined for an integer k and a set of strings. Here is an example of construction for $k = 2$ and strings TGACGTC and TGACTTC. First, we write down all substrings of size k and designate them as vertices:



Afterwards we glue vertices with the same labels to get the graph:



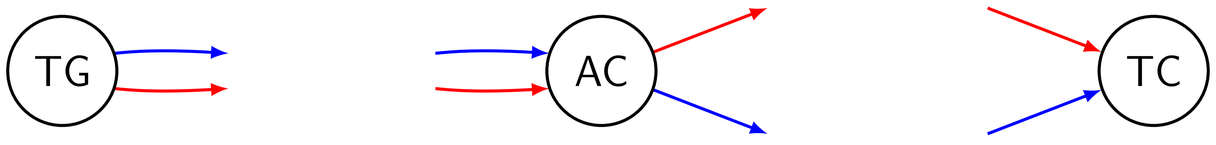
The last step is compression of non-branching paths so that the graph occupies less space:



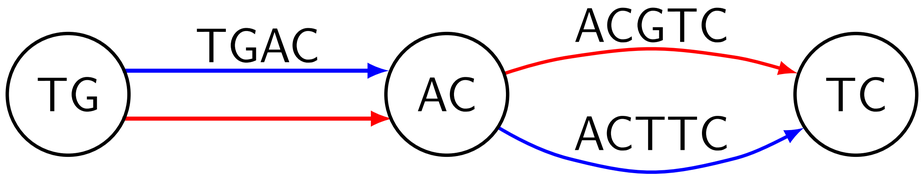
Construction of the compressed graph this way takes many resources. Instead, we construct it directly from the input.

Junctions and Edges

Junctions are vertices that have at least two different predecessors or successors in the graph. First and last k-mers of a string are also junctions:



Junctions are exactly vertices of the compacted graph. Once we know the set of junctions, edges of the compressed graph are trivial to construct. All we have to do is traverse the genomes and record junctions in the order they appear:



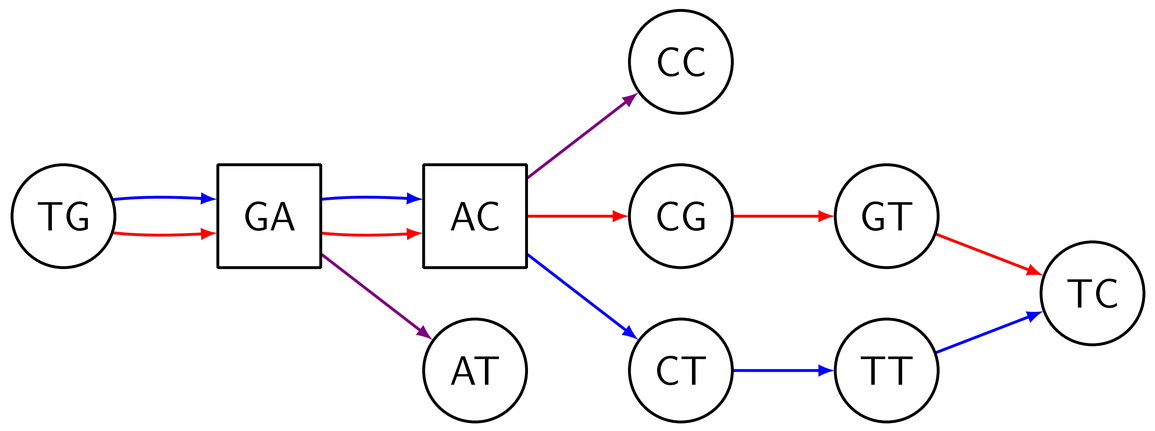
TG GA AC CG GT TC

TG→ AC → TC

This way, we reduce the problem of graph construction to identifying junctions.

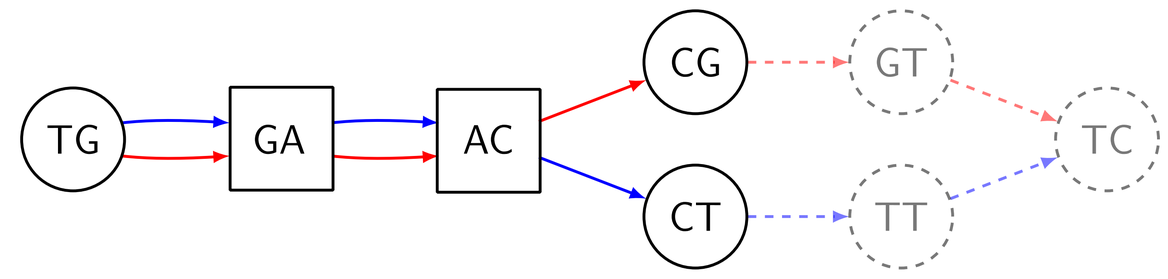
Identification of Junction Candidates

We can narrow down the set of possible junction. To do so, we store edges of the graph in a Bloom filter. Then for each vertex we query its neighbors to identify possible junctions. This representation takes small space, but results in some false junctions due to false positives of the Bloom filter. A picture below shows such representation, violet edges are false positives, squared vertices are junctions candidates:



Refining The Graph

Once we have the junction candidates, we can filter out false positives. To do so, we store edges in a hash table and query neighbors of each vertex. However, we store only edges touching candidates and this way save space, like on the picture below where dashed part of the graph is not stored:



With this representation we can find all the junctions using low memory.

Results

We benchmarked our algorithm against others on both real and simulated data. Below are the results, each cell shows the running time and the memory usage in parenthesis in gigabytes.

Dataset	k	BCALM [4]	Sibelia [5]	SplitMem [3]	`bwt-based` [2]	TwoPaCo [1]	
						1 thread	15 threads
62 *E.coli*	25	6m (1.6)	10m (12.2)	1h (178.0)	8m (0.9)	4m (0.1)	2m (0.4)
	100	13m (2.5)	8m (7.6)	1h (178.0)	8m (0.5)	4m (0.2)	2m (0.4)
7 humans	25	7h (22.4)	-	-	14h (100.3)	7h (4.4)	1h (4.8)
	100	2h (221.7)	-	-	13h (46.0)	5h (8.4)	1h (8.7)
8 primates	25	35h (85.6)	-	-	-	15h (34.4)	2h (34.4)
	100	-	-	-	-	13h (56.1)	2h (61.7)
43+7 humans	25	-	-	-	-	-	11h (69.8)
	100	-	-	-	-	-	15h (70.2)
93+7 humans	25	-	-	-	-	-	23h (77.4)

Discussion

TwoPaCo makes significant progress in extending the number and size of genomes from which a compacted de Bruijn graph can be constructed. We believe that this progress will enable novel biological analyses of mammalian-sized genomes. For example, de Bruijn graphs can now be applied to construct synteny blocks for closely related mammalian species, similar to how they were applied to bacterial genomes. We also developed a procedure that splits input into subsets that can be processed independently. It allows handling large inputs even on machines with limited memory for the cost of running time. TwoPaCo can also be useful in other applications, such as the representation of multiple reference genomes or variants between genomes.

References

[1] Minkin, I., Pham, S., Medvedev, P. (2016). TwoPaCo: An efficient algorithm to build the compacted de Bruijn graph from many complete genomes.
arXiv preprint arXiv:1602.05856.

[2] Baier, U., Beller, T., Ohlebusch, E. (2015).
Graphical pan-genome analysis with compressed suffix trees and the Burrows-Wheeler transform.

[3] Marcus, S., Lee, H., Schatz, M. C. (2014).
SplitMEM: a graphical algorithm for pan-genome analysis with suffix skips.

[4] Chikhi, R., Limasset, A., Jackman, S., Simpson, J. T., Medvedev, P. (2014).
On the representation of de Bruijn graphs.

[5] Minkin, I., Patel, A., Kolmogorov, M., Vyahhi, N., Pham, S. (2013).
Sibelia: a scalable and comprehensive synteny block generation tool for closely related microbial genomes.