A Bruijn Graph Approach

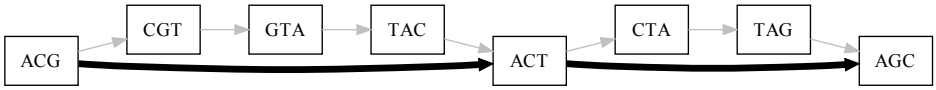
M. Dvorkin, A. Kulikov

May 17, 2011

Contents

1 General Idea

The main goal of this approach is to simplify a *de Bruijn* graph constructed on a given set of reads while still preserving "the structure" of the graph. Namely, instead of representing each read as a sequence of edges between its consecutive k-mers (in which case a read of length r defines $r-k-1$ edges) we represent it as just one (or a few, in general) edge between some of its k-mers. For example, for reads `ACGTACT` and `TACTAGC` and $k = 3$ instead of all gray edges in the figure below we will have only two black edges.



The hope is that the resulting graph will be easier to handle and at the same time it will have essentially the same structure as the original de Bruijn graph.

1.1 Example

Fig. 1 shows de Bruijn (black edges) and A Bruijn (grey edges) graphs for two toy genomes `ACTGACTGTTGACACTG` and `ATTGGTACATTGTGGTACGTACTGACT`. Numbers on edges are their multiplicities. We assume that the genome is circular and that we are given the set of all its reads.

1.2 Advantages of A Bruijn graph over de Bruijn graph

1. Clearly, it requires less memory.

2. In general, we expect that the resulting A Bruijn graph will have the same structure as the corresponding condensed de Bruijn graph. If so, this would allow us to construct the condensed de Bruijn graph using less time and memory.

3. If the used hash-function respects frequency of k-mers the resulting graph will contain a smaller fraction of erroneous k-mers.

Figure 1: De Bruijn (grey) and A Bruijn (black) graphs of genomes `ACTGACTGTTGACACTG` ($readsize = 9$, $k = 5$) and `ATTGGTACATTGTGGTACGTACTGACT` ($readsize = 11$, $k = 5$).

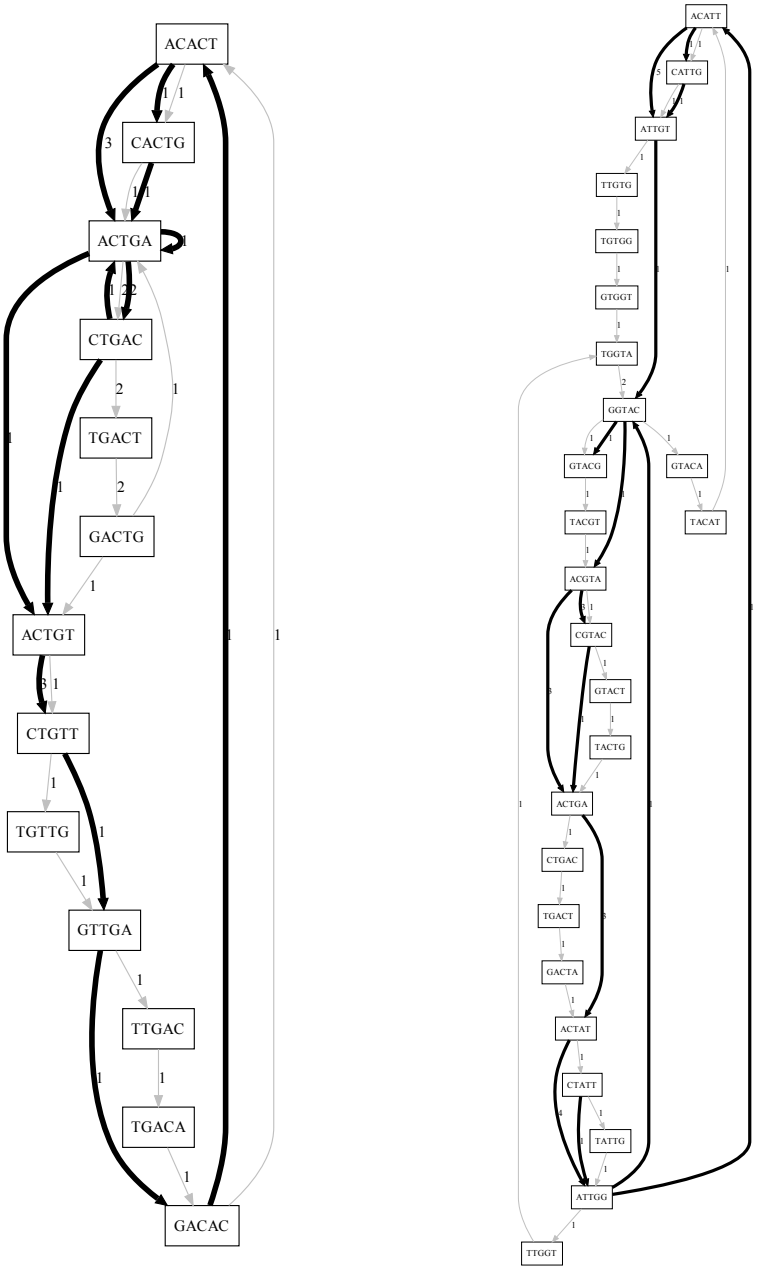
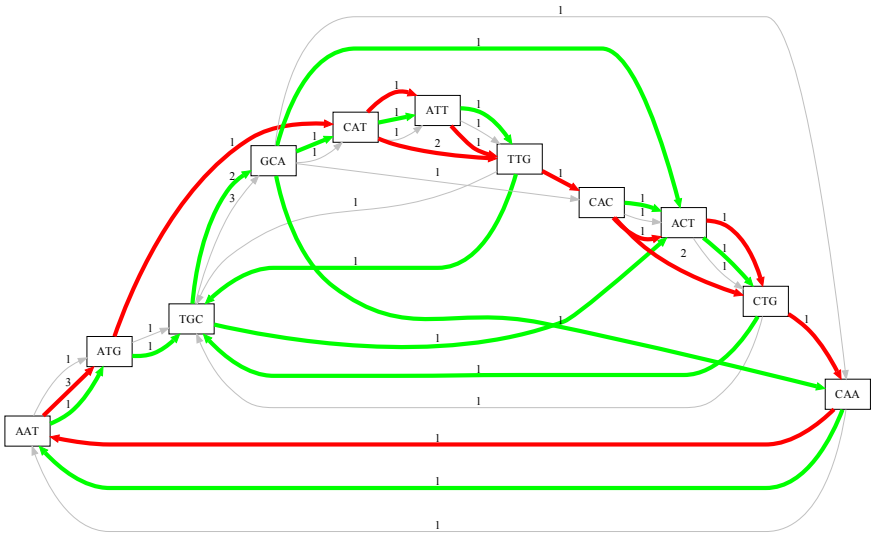


Figure 2: A Bruijn graphs of genome `TAAACGAAAC` ($readsize = 6$, $k = 3$) for different orderings on 3-mers.



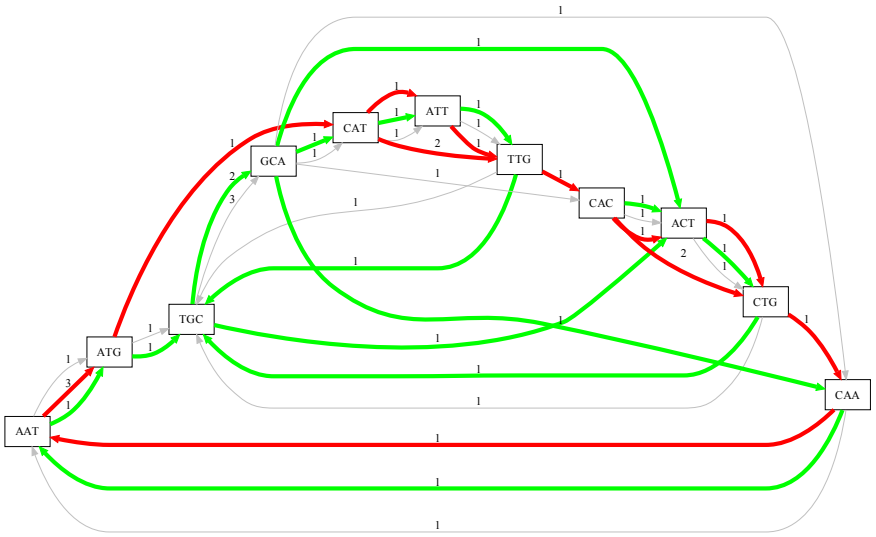
4. If the hash function does not mark k-mers that appear too often, then some of the repeats will be already resolved in A Bruijn graph.

   An example of this effect is given in Fig. 2 where two A Bruijn graphs of a genome `TAAACGAAAC` ($readsize = 6$, $k = 3$) are shown. The difference is in the way 3-mers are ordered. Green edges correspond to the standard lexicographic ordering (`AAA`<`AAC`< $\cdots$ <`TTT`). Now note that the considered genome contains a repeat `AAAC`. Assume that the hash function treats 3-mers `AAA` and `AAC` from this repeat as maximal. Namely, let the ordering be the following:

   $$\texttt{TAA} < \texttt{ACG} < \texttt{CGA} < \texttt{GAA} < \texttt{CTA} < \texttt{ACT} < \texttt{AAA} < \texttt{AAC}\,.$$

   As figure shows, this ordering gives a graph consisting of just one cycle. A slightly more complicated example is given on Fig. 3. There, the corresponding genome `ATGCATTGCACTGCA` contains a repeat `TGCA` three times. Hence de Bruijn graph spells at least two different genomes.

Figure 3: A Bruijn graphs of genome `ATGCATTGCACTGCA` ($readsize = 6$, $k = 3$) for different orderings on 3-mers.



1.3 Hash Functions

One of the possibilities to extract two distinguished k-mers out of a given read is to take two k-mers with minimal value of some hash function h. Some natural properties that h should hold are listed below.

1. The hash function should be easily computed. While iterating through all k-mers of a given read it is also important to have a fast way to recompute the hash value. For this, one may take a kind of polynomial hash function (like in a finger-printing algorithm for the pattern matching problem).

2. It should to be stable with respect to reverse-complementary k-mers, i.e., $h(s) = h(s^{RC})$ so that if we represent a read by an edge (s_1, s_2), then its reverse-complement read is represented by a "reverse" edge (s_2^{RC}, s_1^{RC}). Two natural ways to make out a reverse-complementary stable hash function h out of any hash function h_0 are the following:

$$h(s) = h_0(s) \oplus h_0(s^{RC}) \text{ or } h(s) = \min\{h_0(s), h_0(s^{RC})\}.$$

The main question that still needs to be answered is: does this graph really represent the repeat structure of the genome?

2 Practical results

We've implemented the suggested approach and are currently investigating the resulting A Bruijn graphs on emulated data sets.

The technical issues pushed us towards a two-pass procedure:

1. Each read undergoes a (linear-time) hashing procedure, and two least-valued k-mers are marked as "good". (In fact, not themselves but their hash values are marked. Even if a few extra k-mers become good due to hash collisions, it will not make much harm).

2. For each read, all of its k-mers are checked whether their hash values are good, and the good ones are mapped to corresponding vertices in the future graph. The read is thus counted towards one or more edges. The leavings (in the two ends of the read) are currently not considered.

Figure 4: 10% of E. Coli genome. $K = 31$. Two k-mers from each read.

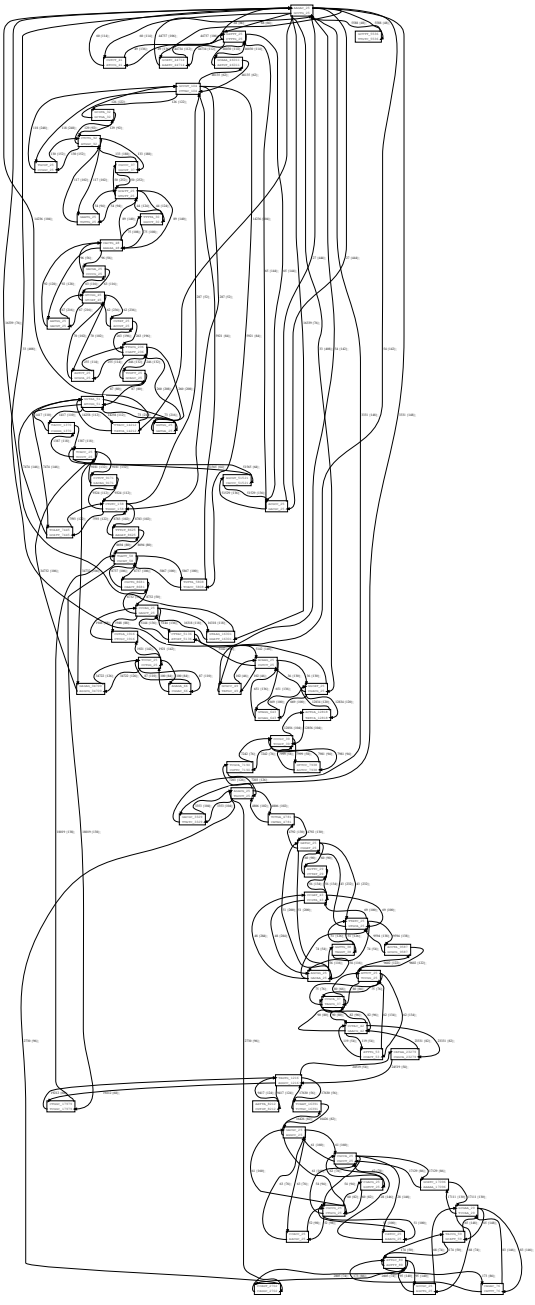
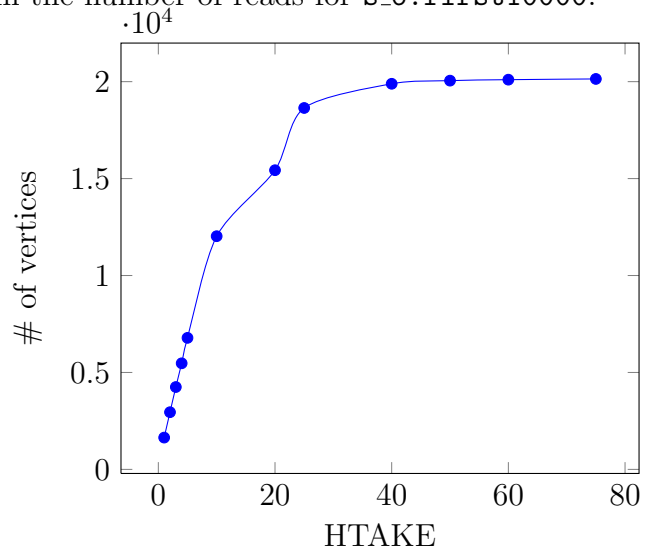


Fig. 4 shows the resulting A Bruijn graph for the first 10% of the reference E. Coli genome; the ideal data set was used for this calculation.

This graph has only undergone a simple vertex-merging procedure; there are more yet-unimplemented simplifications applicable to this graph.

2.1 Statistics

The plot below shows the dependence of the number of vertices in a graph from the number of reads for `s_6.first10000`.



The table below gives the number of vertices in a graph for two extreme values of `HTAKE` (1 and 76) for `s_6.first10000` (program version: Apr 25, 2011; $k = 25$).

	#bp=10000	#bp=100000	#bp=400000
`HTAKE` = 1	1640	15916	62550
`HTAKE` = 76	20144	202838	809730
ratio	≈ 12.2	≈ 12.7	≈ 12.9

3 Further ideas

3.1 Making use of frequency

The task of selecting the k-mers that will become vertices in the A Bruijn graph can be reformulated as the task of selecting criteria that distinguish

some k-mers among others that work the same way in different reads.

Along with a hash function, an apt criterion is the frequency of the given k-mer among all reads (provided the corresponding hash-table is memory-feasible). This criterion might be fruitful considering the following.

- The k-mers that are underrepresented in the reads (even those a with small value of hash function) are (most probably) erroneous. Using frequency allows us to exclude them from the set of vertex-forming k-mers.
- The k-mers that are overrepresented correspond to high-degree vertices in the graph. It might be helpful to exclude them as well in order to make the structure of the graph more feasible. (Motivation: consider two reads, $axyb$ and $cxyd$. A hash function might suggest to select k-mers x and y which would create a fork in the graph. Selecting a, b, c and d on the other hand leads to two separate edges.)

3.2 Reducing the number of vertices

As said in the beginning the main motivation for considering such graphs is that they contain fewer edges than the original de Bruijn graph. Clearly, the number of edges in the A Bruijn graph is also reduced (the vertices with no bold adjacent edges actually do not belong to A Bruijn graph). In order to further reduce the number of vertices we can do the following. First, mark just one k-mer in each read with the minimal possible hash value. If it turned out to be the first or the last k-mer of the current read, mark one more k-mer with the next smallest hash value.

In the ideal case when each possible read is present in the set of reads, this guarantess that at least two k-mers will be marked in each read. In real situations it would be reasonable to take the second k-mer if the first one falls into on of the leftmost or rightmost τ positions for some treshold τ.

3.3 Paired reads

Each single read provides one or several edges with the information of the following kind: the two end-vertices of this edge are likely to be present in the desired graph traversal at the distance exactly d from each other.

Meanwhile, paired reads provide the information of the similar kind, but with the distance known approximately instead of exactly. It is a question

of a separate investigation whether the approximate distances known from paired reads is helpful when traversing an A Bruijn graph.

4 Pseudocode of the assembler

add high-level description of the assembler

Iterative procedure for selecting the set of earmarked k-mers is the following:

1. From each read, select t minimum hash values of its k-mers, add them to the global set of earmarked hash values.
2. (optional, if $t = 1$) For each read that has only one k-mer with earmarked hash value, earmark one more k-mer from this read.
3. In each read, select k-mers that have their hash values earmarked. For corresponding vertices in the resulting A Bruijn graph, add all pairwise edges, storing their lengths.

This procedure results with an A-Bruijn graph that is subject to further processing.

4.1 Tip extension procedure

Due to the fact that not all the k-mers are represented in the A-Bruijn graph, some extra gaps can be unwillingly created. Consider a genome substring $abcde$, where b and d a k-mers that have small hash values and are selected as minimizers. If none of the reads in the input data contain the whole substring bcd, then vertices corresponding to b and d will be tips in the A-Bruijn graph. Meanwhile reads containing abc and cde may be present in the input data, thus it is an error of the approach to call this vertices tips and to introduce a gap here.

The tip extension procedure is suggested to handle this issue. It consists of three steps that are repeated consequently until no possible tip extensions are present. (Each steps demands reading the entire input data. It is possible to reduce the procedure to less than three steps, but much more memory will be used in that case).

1. By processing all the reads, find out for each earmarked k-mer, whether we've seen any earmarked k-mer to the left from it, and to the right. Those earmarked k-mers that don't have either left or right "neighbours" (or both) are called tips.

2. By processing all the reads, find the collection of all the non-earmarked k-mers that are present (at least once) in the same read with any tip. Call them possible tip extensions.

3. By processing all the reads, for each possible tip extension, find whether there was any earmarked k-mers to the left from it, and to the right.

Now that this information is collected, each tip is being extended with a possible tip extension of the following type (in the order preference):

1. A possible tip extension that was just earmarked (in this iteration) as an extension of another tip. In this case nothing is to be done, the tip already became a non-tip vertex.

2. A possible tip extension that has both left and right neighbours. This extension becomes earmarked and "kills" one undesirable gap.

3. Any possible tip extension. Among these, the most distant from the tip (in nucleotides) is selected, it becomes earmarked and extends the tip as far as possible. This doesn't resolve a gap, but allows not to lose information close to an actual gap (or the end of an actual scaffold).

If no new earmarked k-mers were introduced after this procedure, stop, otherwise repeat the entire procedure.