Seminar paper on: Read mapping on de Bruijn graphs

Jouko Niinimäki

Seminar paper
UNIVERSITY OF HELSINKI
Department of Computer Science

Helsinki, December 2, 2016

HELSINGIN YLIOPISTO — HELSINGFORS UNIVERSITET — UNIVERSITY OF HELSINKI

Tiedekunta — Fakultet — Faculty	Laitos — Institution — Department
Faculty of Science	Department of Computer Science

Tekijä — Författare — Author
Jouko Niinimäki

Työn nimi — Arbetets titel — Title

Seminar paper on: Read mapping on de Bruijn graphs

Oppiaine — Läroämne — Subject
Computer Science

Työn laji — Arbetets art — Level	Aika — Datum — Month and year	Sivumäärä — Sidoantal — Number of pages
Seminar paper	December 2, 2016	9

Tiivistelmä — Referat — Abstract

Sequencing and aligning the resulting reads on reference contigs is a common task in the field of bioinformatics. Still, there are no consistently good software to perform the alignment well in different situations. An idea is represented here to use a de Bruijn graph generated from the reads as a reference instead of contigs. Tests on generated and real data implies that with this method both the alignment quality and computational resource usage are improved.

ACM Computing Classification System (CCS):
Theory of computation → Design and analysis of algorithms → Data structures design and analysis → Pattern matching
Applied computing → Life and medical sciences → Computational biology → Molecular sequence analysis

Avainsanat — Nyckelord — Keywords
read mapping, de Bruijn graph, bioinformatics

Säilytyspaikka — Förvaringsställe — Where deposited

Muita tietoja — Övriga uppgifter — Additional information

Contents

1 Introduction **1**

2 De Bruijn Graph **3**

3 The Algorithm **4**

4 Results **6**

5 Conclusion **8**

References **9**

1 Introduction

One frequently permormed task in the field of bioinformatics is sequencing a genome of some organism. That is, finding out the DNA sequence of the individual. The state-of-the-art technologies cannot yet produce full DNA sequence from a species but instead short *reads*, subsequences of length of a few hundred bases (A, C, G or T). The sequencing machines produce millions or even billions of these reads from random positions of the sequence without any location information[1]. A full DNA sequence must be then composed from these reads. This procedure is called read mapping and can be done *de novo* (without any reference) by using the overlapping parts of the reads to sort them, or by using another sequence, for example of a closely related species, as a reference. In some cases the reference is not a full sequence but a set of shorter subsequences, so called *contigs*, which often are results of de novo assembly, as the assembly methods are rarely capable of constructing a full sequence, but multiple contigous subsequences instead. In this article we consider only the case of reference contigs, or more accurately, the case where the reads of the reference sequence are available.

As the sequences are usually huge (from some microbe's several millions to for example human's over three billion bases [RUC+11, pages 275, 478]) and so is the number of the reads, the mapping process is a time consuming task and requires a lot of computational power. In practice the mapping is done by *aligning* the reads against the reference sequence in the best possible way. Aligning means positioning sequences next to each other in a way that shows the assumed relation of the sequences. To find the best possible alignment one needs a way to evaluate the result. One commonly used way is to calculate a score based on the matches and mismatches in the alignment. Example alignments along with their scores calculated by assigning +5 for the matching positions and -4 for the mismatching positions are shown in Figure 1.1.

Finding the optimal alignment of two sequences (using dynamic programming) takes $O(nm)$ time, where n and m are the lengths of the sequences [SW81]. As the length of the reference sequence and the total length of all

[1]Based on https://www.illumina.com/, 10.11.2016

the reads to be aligned can be huge, numerous faster heuristic algorithms have been created to reduce the computation time [LH10]. One group of these algorithms is based on creating a hash table from each k-mer, a subsequence of length k, and their positions in one of the sequences. The other sequence is then scanned k-mer by k-mer to find matching positions in the hash table. The possible findings, *hits*, are then extended in both direction using previously mentioned scoring system (and possibly combined together with other close hits) to get the final alignments. The principles of the algorithm are illustrated in Figure 1.2. Another commonly used heuristic way to find alignments is based on suffix-arrays (or trees) with effective use of Burrows-Wheeler transformation.

Although reasonably fast, these algorithms and mapping techniques on contigs are far from perfect. One idea to improve the mapping quality is to use a graph as a reference instead of contigs. One way to create contigs is based on constructing a graph from the reads and extracting them from the paths of the graph that seem continuous. Resulting contigs don't however necessarily represent the real sequence, some of them share similar parts and some of them are often abandoned. One idea of aligning the reads to the graph instead of contigs is to reduce the problems that aligning on the contigs have. A practical algorithm based on de Bruijn graph (and a software called BGREAT utilizing the idea) is presented in next two chapters. Results and analysis are shown in the latter part of the paper. This seminar paper is mostly based on [LCRP15] and therefore all the noncited pieces of information (including tables and algorithms) can be aasumed to be based on that paper.

Figure 1.1: An example alignment and calculation of it's score.

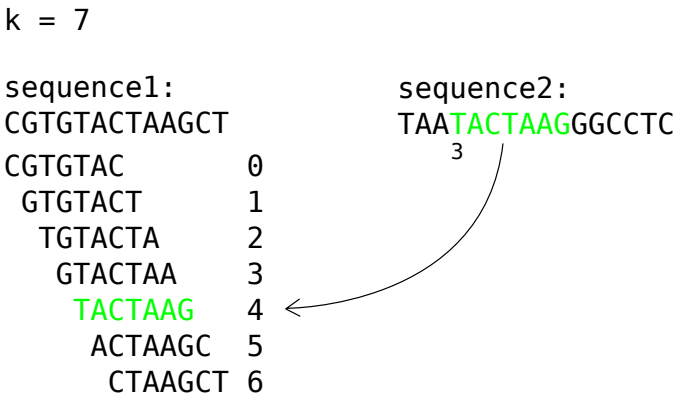

Figure 1.2: An example of locating hits between two sequences. A hash table is created for each k-mer and their positions on sequence 1. Sequence 2 is scanned through to find same k-mers (using each k-mer to query the hash table).

2 De Bruijn Graph

De Bruijn graph of order k for sequence S is a directed graph where each different k-mer in the sequence is represented as a node in the graph. Two nodes v and u are connected with an edge from v to u if they are found consecutively v before u in sequence S. The formal definition can be seen below:

Definition 1. De Bruijn graph (V, E) of order k for sequence S is a direceted graph iff:
a) Each unique k-mer in S is a node in v and there are no other nodes
b) Two nodes u and v have an edge from u to v iff k-mer u found in $S[i...i+k]$ overlaps k-mer v so that v is found in $S[i+1...i+k+1]$

The original sequence can be obtained from the graph by first initializing the (empty) sequence with k-mer from the start node and then following a certain path from the start node to the end node while appending the last letter from each node to the sequence. As we can clearly see, there can also be another paths through the graph, so some of the sequence information is lost while constructing the graph.

As we notice, the total length of De Bruijn graph can be at most $n-k+1$ nodes (n is the length of sequence), with k letters on each node. The size of the graph can be reduced without losing any information by using *compacted* de Bruijn graph. Compacted de Bruijn graph can be modified from de Bruijn graph by combining sequential nodes that have only one ingoing and outgoing edge to form a larger node with length $k+c-1$, where c is the number of the nodes combined. These larger nodes are called *unitigs*.

Each edge (u, v) in the graph can be considered to represent the shared part of the two nodes u and v, that is a $(k-1)$-mer. These $(k-1)$-mers are called *overlaps*. Also, as the DNA alphabet consists only of four characters, each node can have at most four incoming and four outgoing edges (overlaps). Each incoming edge to a single node represents the same overlap with each other as do also the outgoing ones.

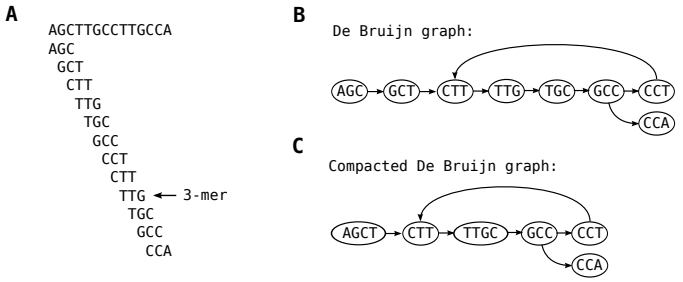

Figure 2.1: An example image illustrating de Bruijn graph. **A**: a sequence and it's 3-mers. **B**: a de Bruijn graph made out of sequence in A. **C**: a compacted de Bruijn graph of the same sequence.

3 The Algorithm

The algorithm can be divided into two phases: (**1**) aligning the reads on unitigs of a compacted de Bruijn graph created from the reads and (**2**) aligning them on the whole graph using overlaps. In step (**1**), for each unitig larger than a read, the read is aligned against the unitig using traditional methods. In BGREAT, a software created by the authors, this is done with external software, Bowtie2 [LS12]. The alignment is done as if the unitig

was a single target sequence. In step (**2**) a hash table is created from the graph's overlaps using overlaps ($(k-1)$-mer) as keys and their position in the graph as values. A read is then scanned $(k-1)$-mer by $(k-1)$-mer trying to find hits in the hash table. If a single hit is found, the alignment is tried to be extended over the unitigs on both sides of the overlap. If multiple hits are found, the algorithm tries to find a suitable path from the graph that matches the hits and the read is aligned against the unitigs found along the path. In both cases the alignment's score has to reach a given threshold to be accepted.

The second part is next explained in more detail. For the first t (in the original paper $t = 2$) hits in a read, each pair of unitigs (connected by the overlaps that caused the hit) is aligned on the reads on the position that the hit suggests. The same is done with the last t hits. The best alignments are picked as a beginning and an ending of a full path. Once this is done, the algorithm greedily tries to match the next four possible unitigs implied by the next hit, until there are no hits (and the beginning and the ending have been connected). To speed up the process, if one of the unitigs ends with the overlap that is next in line in the hit queue that unitig is tested first and accepted straight if the resulting alignment's score is within a given error threshold. The psedocode of the second part of the algorithm can be seen in Algorithm 1.

The beginning and the end of the path are done separately to reduce unnecessary mapping calculation for reads that cannot be mapped. Also this is done for the first t hits to prevent possible errors in the overlaps from effecting the result.

The algorithm fails to find an alignment (a path) in three cases: (**1**) if all the overlaps that should be found in the read contain errors, (**2**) the first or last t overlaps and their unitigs won't map (meaning the hits are false positive) or (**3**) the greedy algorithm makes wrong choices and therefore the beginning and the end is not connected.

Data: input: read r, int t
for *t first overlaps of r* **do**
    Find a path *beg* that map the begin of r
    if *beg found* **then**
        for *t last overlaps of r* **do**
            Find a path *end* that maps end of r
            if *end found* **then**
                Find in a greedy way a path *cover* that maps the read from *beg* to *end*
                if *cover found* **then**
                    write path
                    return
    Algorithm 1: Mapping a read on unitigs of de Bruijn graph

4 Results

To test the mapping quality of BGREAT four sets of reads with different complexities were used first to generate a compacted de Bruijn graph. The graph was used to create contigs with external tools, Velvet and Minia [ZB08][CR13]. Then the reads were mapped back on the de Bruijn graph using BGREAT. For comparison, the reads were also mapped on the contigs with another widely used mapping software, Bowtie2, the same software that BGREAT uses on mapping the reads on unitigs.

On all four different data sets BGREAT was able to map back the reads significantly better (from 1.6 to over 20 percent units) than Bowtie. The mapping percentages can be seen in Table 4.1. As seen in the table, the gain gets clearly larger when the complexity of the graph rises.

	Reads mapped on	
Reads from	contigs	graph
`E.coli`	95.57 %	97.16 %
`C.elegans`	80.60 %	93.24 %
`C.elegans complex`	56.33 %	89.15 %
`H.sapiens`	63.16 %	85.70 %

Table 4.1: Percentages of reads that were able to map back to contigs (first column) or graph (second column) created from the reads. Three different set were used, one of which was used to create also more complex graph (and therefore also contigs?).

In previous tests the same reads were used to construct the de Bruijn graph and then mapped back on it. In real life, however, the purpose is to map a different individual or species against another and therefore some differences (caused by mutations) exists between the the graph and the reads. To test this, another set of reads of *C.elegans* was mapped on the previously constructed graph resulting in 89 % reads mapped with 15 % of the reads mapped on the second phase (on the branching paths).

Further tests were maid with following simulated data created from human chromosome 1: Six set of reads were artificially generated from the sequence with difference rate of 0, 0.1, 0.2, 0.5, 1.0 and 2.0 percents of the original. They were then mapped on the de Bruijn graph created from the original sequence. The quality of the alignments (of reads that were mapped in right places) were then calculated using the number of mismatches as the distance from original. The results can be seen in Table 4.2. As we can see from the table, the quality is relatively good even with the highest difference rate (99,9 % of the reads were mapped with at most 3 mismatches).

Difference percent	Percent of reads that were mapped within distance of			
	0	1	2	3
0	100	100	100	100
0.1	99.31	99.83	99.92	99.96
0.2	98.79	99.70	99.91	99.98
0.5	97.20	99.37	99.78	99.95
1	94.88	98.60	99.52	99.93
2	90.85	97.28	99.07	99.90

Table 4.2: Running time and peak memory usages of four different read sets using BGREAT (first column) and Bowtie2 (second column). Read sets are not comparable to each other due to different sizes (except with *C.elegans*).

The performance differences between the two mapping techniques are shown in Table 4.3. Based on these results, BGREAT seem to be generally faster than Bowtie and also consume significantly less memory while also being able to reach better mapping percentages as pointed out previously. The only case where BGREAT performs slower, is the complex graph of *C.elegans*. However, this is also the case where the mapping percentage was

most improved (from Minia+Bowtie's 53 % to BGREAT's 85 %).

While de novo assembly of the contigs is actually a separate task of mapping on the contigs, it's reasonable to point out that the peak memory consumption performing this task on *C.elegans*' set of reads with Velvet was over 80GB, while the BGREAT's memory consumtion peaked only at 4GB.

	BGREAT		Bowtie2 (on contigs)	
Reads from	CPU time	memory	CPU time	memory
`E.coli`	1m 40s	19 MB	3m 53s	29 MB
`C.elegans`	72m 31s	975 MB	33m	1.66 GB
`C.elegans complex`	51m 28s	336 MB	72m 31s	493 MB
`H.sapiens`	87h	9.7 GB	90h 15m	21 GB

Table 4.3: Running time and peak memory usages of four different read sets using BGREAT (first column) and Bowtie2 (second column). Read sets are not comparable to each other due to different sizes (except with *C.elegans*).

5 Conclusion

Mapping reads on reference contigs is a time and performance consuming operation, that still has sapce to be improved. Using de Bruijn graph, which is already sometimes used in constructing the contigs, instead of the contigs as a reference can result significantly better mapping percentage with good quality and less usage of computational resources. There are, however, situations in which this method may fail, especially if the overlaps of the graph are faulty. As the improved mapping relies heavily on these overlaps, the issue is something to be looked at. Still, as the tests imply, even with these flaws the the method surpasses the traditional way of mapping reads on contigs in almost every aspect.

References

[CR13] Chikhi, Rayan and Rizk, Guillaume: *Space-efficient and exact de bruijn graph representation based on a bloom filter*. Algorithms for Molecular Biology, 8(1):1, 2013.

[LCRP15] Limaset, Antoine, Cazaux, Bastien, Rivals, Eric, and Peterlongo, Pierre: *Read mapping on de bruijn graphs*. arXiv preprint arXiv:1505.04911, 2015.

[LH10] Li, Heng and Homer, Nils: *A survey of sequence alignment algorithms for next-generation sequencing*. Briefings in Bioinformatics, 11(5):473–483, 2010.

[LS12] Langmead, Ben and Salzberg, Steven L: *Fast gapped-read alignment with bowtie 2*. Nature methods, 9(4):357–359, 2012.

[RUC+11] Reece, Jane B, Urry, Lisa A, Cain, Michael L, Wasserman, Steven A, Minorsky, Peter V, and Jackson, Robert B: *Campbell Biology*. Pearson Education, 9. edition, 2011.

[SW81] Smith, Temple F and Waterman, Michael S: *Identification of common molecular subsequences*. Journal of Molecular Biology, 147(1):195–197, 1981.

[ZB08] Zerbino, Daniel R and Birney, Ewan: *Velvet: algorithms for de novo short read assembly using de bruijn graphs*. Genome research, 18(5):821–829, 2008.