2012 International Conference on Bioscience, Biochemistry and Bioinformatics
IPCBEE vol.31(2012) © (2012)IACSIT Press, Singapoore

An Algorithm for Classifying DNA Reads

Mohammed Sahli [1], Tetsuo Shibuya [2]

[1] Department of Computer Science, Graduate School of Information Science and Technology, University of Tokyo - 7-3-1 Hongo, Bunkyo-ku, Tokyo 113-0033, Japan

[2] Human Genome Center, Institute of Medical Science, University of Tokyo - 4-6-1 Shirokanedai, Minato-ku, Tokyo 108-8639, Japan

Abstract. A DNA read comes from either strand of the genome, and it has been believed in the impossibility of determining which one is the actual strand for the last three decades. Therefore, for the first time, we developed an algorithm for classifying DNA reads according to which strand they were generated from. Consequently, misassemblies can be reduced by integrating this algorithm as part of the general genome assembling pipeline. Moreover, the genome assembly problem can be further simplified by building an assembler which this algorithm is involved in. We validated this algorithm by comparing some existing assemblers with an assembler that uses our algorithm The DNAClassifier tool, that uses only this algorithm, was developed so that it can be pipelined with other tools in the genome assembly process. All versions of our assembler as well as the DNAClassifier tool are both available for free to the public and can be downloaded from http://sourceforge.net/projects/dnascissor/files/.

Keywords: genome assembler, reads classification, short reads, shotgun sequences.

1. Introduction

The genome assembly problem is a challenging problem in Bioinformatics. It consists in bringing millions of genomic reads all together in order to reconstruct the original sequence from which they were randomly originated from. Sequencing errors, repeats and reads orientation are the main constraints of this problem. Due to the difficulty of these constraints, the current assemblers are not able to reconstruct the genome but a set of contiguous sequences (i.e. contigs or scaffolds). Two approaches have been studied extensively in the literature: the Overlap-Layout-Consensus approach and the de Bruijn graph-based approach. The first approach consists in computing the overlaps between all genomic reads and constructing the so-called "the overlap graph". Some assemblers that use this approach include: PCAP [7], TIGR [10], CAP3 [6], the string graph of Myers [11] and MIRA [12]. However, computing the overlaps is the most time-consuming task and the main drawback of this approach. In addition, dealing with repetitive reads is quite difficult in the overlap graph. Consequently, in order to overcome these weaknesses, a fundamentally different approach was adopted by Pevzner et al. [2]. In this approach, sequences are cut into smaller reads of the same length k (generally called k-mers) and mapped as paths through the graph (i.e. de Bruijn graph). Some of the assemblers that use this approach are: Euler assembler [2], SSAKE [13], EULER-SR [3], Velvet [8], ALLPATHS [14], ABYSS [15], and SOAPdenovo [9]. It has been shown that the de Bruijn graph-based assemblers are more effective in practice for short read assemblies. However, despite the advantages of the de Bruijn graph data structure, it has not been analyzed deeply in current assembly methods. We developed an algorithm that traverses this graph in order to classify DNA reads depending on where they came from. We give the details of this algorithm so that it can be used by researchers for future assemblers. We developed a de novo assembler (Arapan-M) that integrates this algorithm. The DNAClassifier is another tool that we developed and it uses only the classification algorithm so that it can be easily pipelined in the assembly process of other assemblers.

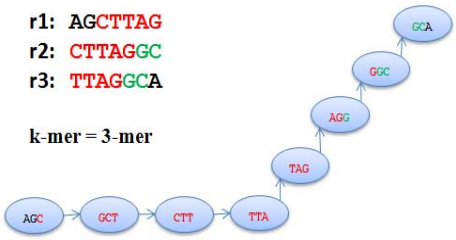

Figure 1. A simple path constructed from three genomic reads (based on the de Bruijn graph).

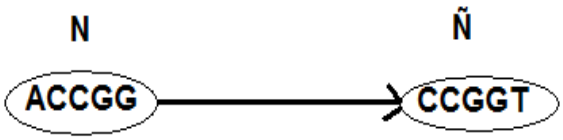

Figure 2. The edge that goes out of the node *N* to its opposite node *Ñ* is considered as a false connection.

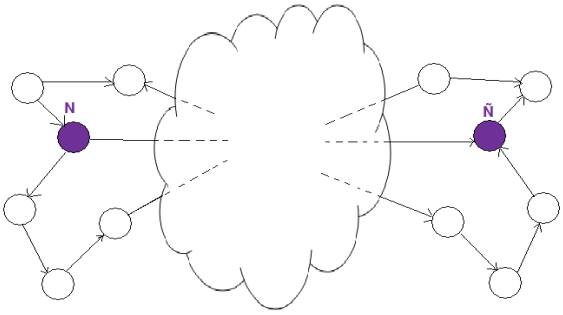

Figure 3. The algorithm creates two sets by starting at an arbitrary node *N* and its opposite node *Ñ*.

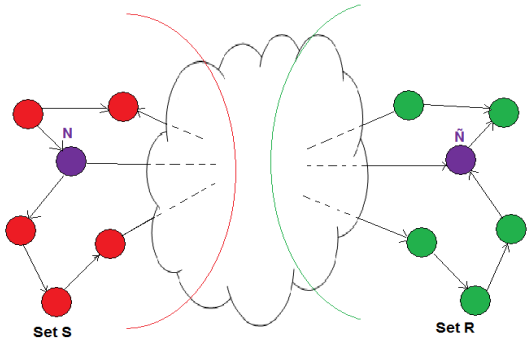

Figure 4. The algorithm stops when there is no further exploration. Set *S* is connected with set *R* through direct connections.

2. Methods

2.1. De Bruijn sequences

Let *S* be a string of length *n*, the *k*-mer composition or spectrum of *S* is the multiset of $n-k+1$ *k*-mers in *S* and is written *Spectrum*(*S,k*) [1]. Let $k=4$ and *S*=ACGTTGTG; then *Spectrum*(*S*,4)={ACGT, CGTT, GTTG, TTGT, TGTG}. This idea is used in the de novo genome assemblers and it was originally introduced in [2]. We consider a set of DNA reads which were generated from the same genome. In order to reconstruct the original sequence, we cut the reads into even smaller *k*-mers of the same length *k*. We say two *k*-mers *p* and *q* overlap if $overlap(p,q)=k-1$, that is, the suffix of length $k-1$ of *p* coincides with the prefix of length $k-1$ of *q* [1]. Concatenating all the *k*-mers leads to a graph structure called the de Bruijn Graph (Figure 1). This graph, which became the pillar of the de novo genome assemblers, can be used to solve the problem of repeats more effectively than the overlap graph.

2.2. Basic idea for classifying reads

This paper suggests an approach to the DNA reads classification problem based on the notion of the de Bruijn graph. In this graph, each node *N* represents a *k*-mer and has an opposite node *Ñ* in the graph. The node *Ñ* is characterized by the reverse compliment *k*-mer of that of the node *N*. The constructed graph may have *p* connected components. The main idea underlying this algorithm is to create two sets of nodes for each connected component. Each set will represent the *k*-mers which were generated from the same strand. This task is done by traversing the corresponding connected component starting at an arbitrary node to create the first set (i.e. the first strand) whereas the second set (i.e. the reverse strand) is being simultaneously created starting at the opposite node *Ñ*. The details of the algorithm are given below.

2.3. The algorithm

Once the graph is created, we identify its connected components by running the depth-first search algorithm. If at least one node and its opposite node exist in the same connected component, it will be declared as a false component. For each false connected component, we do the following operations:

A) Light isolation: An edge is identified as a false connection if it goes out of a node *N* to its opposite node Ñ. This happens frequently in the case of an even length palindrome as illustrated in Figure 2. All false edges are eliminated in this stage. The pseudo-code of this procedure is given below.

```
Algorithm: light_Isolation(Graph G(V,E), Set M)
Input: V : set of nodes;
    E : set of edges;
    M: set of the couple "node and its opposite"
Begin
for x=(a,b)∈M do
        if (a,b)∈E then
                E:=E\{(a,b)}
End.
```

B) Hard isolation: Let *S* and *R* be two sets. We say there is a *direct connection* between *S* and *R* if and only if there is an edge that goes out of a node in one set and points to another node in the other set. Initially, only a single start node *N* and its opposite node $\tilde{N}$ are marked "*discovered*". *S* and *R* are initialized by *N* and $\tilde{N}$ respectively (Figure 3). To completely explore a node, we must evaluate edges going out of it and edges that point to it. If an edge goes to undiscovered node, we mark it and its opposite "*discovered*" and add them to *S* and *R* respectively. If an edge goes to a *discovered* vertex, it will be ignored because further contemplation will tell us nothing new about the component. The process stops exploring from a discovered node only if it has a *direct connection* to the other set (Figure 4). When there is no further exploration, we will only get a set of edges that go out of a set and point to another. These edges are simply removed, and as a result the nodes will not share the same connected component that contains their opposite nodes. Consequently, each false connected component will be divided into two subcomponents. The pseudo-code of the algorithm is given below.

```
Algorithm: hard_Isolation(Component C(V,E) , Set M)
Input: V : set of nodes;
       E : set of edges;
       M: set of the couple "node and its opposite"
Output: Component C(V,E);

Set  T, R, Q, S;
Integers  i, a, b;

1.  Begin
2.  //Step 1: Initialization
3.  Let a and b be two nodes such that a,b∈V and (a,b)∈M;
4.  V:=V\{a,b};
5.  Q:={x / (a,x) ∈E};
6.  Q:=Q∩{y / (y,a)∈E};
7.
8.  //Step 2:Determine the two subcomponents
9.  while V≠∅ and Q≠∅ do
10.   for all x∈Q do
11.     let y∈V such that (x,y)∈M;
12.     if y∉N then
13.        T:=T∪{x};
14.        R:=R∪{y};
15.        S:=S∩{z / (x,z) ∈E};
16.        S:=S∩{z / (z,x) ∈E};
17.     else
18.        S:=S∩{z / (y,z) ∈E};
19.        S:=S∩{z / (z,y) ∈E};
20.     end for.
21.     for all x∈Q do
22.       V:=V\{x,y} such that (x,y)∈M;
23.     end for.
24.     Q:=S;
25.     S:=∅;
26. end while.
27.
28. //Step 3: remove all edges between T and R
29. for all x∈T do
30.   for all y∈R do
31.     if (x,y)∈E then
32.       E:=E\{(x,y)};
33.   end for
34. end for
35. End
```

Table 1. The short read datasets used for comparison

Reference genome (ID)	Accession no.	Size (bp)	Read len. (bp)	Coverage	Error rate (%)	Type of dataset
Beta vulgaris genomic clone ZR-47B15 (D1)	ZR-47B15	117150	27	616	2.10	Real
E. coli str.*K*-12 substr. MG1655 (D2)	NC_000913	4639675	100	35	1.0	Simulated
Saccharomyces cerevisiae, Chr. 5 (D3)	NC_001137	576874	100	35	1.0	Simulated
Saccharomyces cerevisiae, Chr. 7 (D4)	NC_001139	1090946	70	35	3.0	Simulated
Haemophilus influenzae (D5)	NC_007146	1914490	100	35	1.0	Simulated

Table 2. The comparison

Genome	Assember	No. of contigs	N50	Average length	Max length	Total length	Genome coverage
D1	Arapan-M	69	2147	1581	8449	109124	93.15%
	Velvet	75	1912	1446	9108	108495	92.61%
	ABYSS	68	1900	1576	7088	107211	91.52%
D2	Arapan-M	898	8249	4969	35440	4462975	96.19%
	Velvet	1083	6491	4139	41201	4483587	96.64%
	ABYSS	931	8113	4795	35440	4464731	96.23%
D3	Arapan-M	80	15293	6794	41953	543540	94.22%
	Velvet	109	8777	5060	25051	551618	95.62%
	ABYSS	92	15293	5991	41953	551233	95.56%
D4	Arapan-M	168	11129	6184	32711	1039034	95.24%
	Velvet	235	8210	4463	36131	1048944	96.15%
	ABYSS	186	10201	5657	32711	1052327	96.46%
D5	Arapan-M	661	3787	2777	20265	1835788	95.89%
	Velvet	634	3768	2900	21502	1838987	96.06%
	ABYSS	688	3564	2664	20265	1833402	95.76%

3. Result

We developed the DNAClassifier tool to classify DNA reads depending on their strands and outputs a set of *k*-mers (that came from the same strand) in FASTA format. It was developed so that it can be pipelined in the assembly process. Moreover, we also integrated the DNA reads classification algorithm as a main phase in our de novo assembler (Arapan-M). All datasets which were used for making the comparison are shown in Table 1. D1 was downloaded from sharcgs.molgen.mpg.de/download.shtml and it was initially used in [4][5]. All the experiments were run on an Intel(R) Core(TM) i7 CPU computer with eight cores operating at 2.93 GHz, 11.8 GB of memory, and Kernel Linux 2.6.32-64-generic operating system (Ubuntu). DNAClassifier and Arapan-M were implemented in C/C++ and compiled and run with the aid of a cross-platform application framework called Qt version 4.6.2 (64 bit). We compared the result of our assembler with ABySS and Velvet 1.1.3. The results are shown in Table 2. Arapan-M always achieved fewer contigs and higher N50 length compared to other assemblers. Although Velvet achieved the best maximum length of contigs, it produces great number of contigs compared to other assemblers. ABySS was the most competitive to Arapan-M in most cases.

4. Discussion

It was believed in the impossibility of solving the DNA reads classification problem. However, for the first time, we introduced an algorithm for tackling this problem. We also developed two tools that used this algorithm to illustrate its utilization and compared the results with other de novo assemblers. Our assembler can output the de Bruijn graph in different formats. Figure A and Figure B (see the appendix) shows the graph diagram of the *white tailed deer coronvirus genome*. These graphs were generated by another version of our assembler (Arapan-S). The two strands of the genome can be seen clearly in the graphs. However, this is only the case of a tiny genome, but for medium and big size genomes, the graphs will consist of a set of connected components. Each of these has its opposite in the graph. This is because of sequencing errors that limit the robustness of the DNA reads classification problem.

5. Acknowledgements

This work was partially supported by the Grant-in-Aid from the Ministry of Education, Culture, Sports, Science and Technology of Japan. We are grateful to Pr. Satoru Miyano (the head of Laboratory of DNA

Sequence Analysis and Laboratory of Sequence Analysis, Human Genome Center, University of Tokyo) for his additional support in publishing this work. We thank Mr. Yassine Bouhmadi for his correction.

6. References

[1] N.C. Jones and A. Pevzner. An Introduction to Bioinformatics Algorithms. *MIT Press.* 2000, Chapter 8, page 268.

[2] P.A. Pevzner, H. Tang, M.S. Waterman. An Eulerian path approach to DNA fragment assembly. *Proc Natl Acad Sci.* 2001, **98**: 9748–9753.

[3] M.J. Chaisson and P.A. Pevzner. Short read fragment assembly of bacterial genomes. Genome Res. 2008, **18**: 324–330.

[4] J.C. Dohm, C. Lottaz, T. Borodina, H. Himmelbauer. SHARCGS, a fast and highly accurate short-read assembly algorithm for de novo genomic sequencing. *Genome Research*. 2007, **17**: 1697-1706.

[5] J.C. Dohm, C. Lottaz, T. Borodina, H. Himmelbauer. Substantial biases in ultra-short read data sets from high-throughput DNA sequencing. *Nucleic Acids Res*. 2008.

[6] X. Huang. and A. Madan. CAP3: A DNA sequence assembly program. *Genome Res*. 1999, **9**: 868–877.

[7] X. Huang, J. Wang, S. Aluru, S.P. Yang, L. Hillier. PCAP: A whole-genome assembly program. *Genome Res.* 2003, 13: 2164–2170.

[8] D.R. Zerbino and E. Birney. Velvet: Algorithms for de novo short read assembly using de Bruijn graphs. *Genome Res.* 2008, **18**: 821–829.

[9] R. Li, H. Zhu, J. Ruan, W. Qian, X. Fang, Z. Shi, Y. Li, S. Li, G. Shan, K. Kristiansen, et al. De novo assembly of human genomes with massively parallel short read sequencing. *Genome Res.* 2010, **20**:265–272.

[10] G.G. Sutton, O. White, M.D. Adams, and A.R Kerlavage. TIGR Assembler: A new tool for assembling large shotgun sequencing projects. *Genome Sci. Technol.* 1995, **1**: 9–19.

[11] E.W. Myers. The fragment assembly string graph. *Bioinformatics.* 2005, **21**: ii79–ii85.

[12] B. Chevreux, T. Pfisterer, B. Drescher, A.J. Driesel, W.E.G. Müller, T. Wetter, and S. Suhai. Using the miraEST Assembler for Reliable and Automated mRNA Transcript Assembly and SNP Detection in Sequenced ESTs. *Genome Res*. 2004, **14**: 1147-1159.

[13] R.L. Warren, G.G. Sutton, S.J. Jones, and R.A. Holt. Assembling millions of short DNA sequences using SSAKE. *Bioinformatics.* 2007, **23**: 500–501.

[14] J. Butler, I. MacCallum, M. Kleber, I.A. Shlyakhter, M.K. Belmonte, E.S. Lander, C. Nusbaum, D.B. Jaffe ALLPATHS: De novo assembly of whole genome shotgun microreads. *Genome Res.* 2008,**18**: 810–820.

[15] J.T. Simpson, K. Wong, S.D. Jackman, J.E. Schein, S.J. Jones, I. Birol. ABySS: A parallel assembler for short read sequence data. *Genome Res.* 2009, **19**:1117–1123.

Appendix

Figure A: The resulted graph of white tailed deer coronvirus genome by GraphViz visualization tool (sfdp layout)

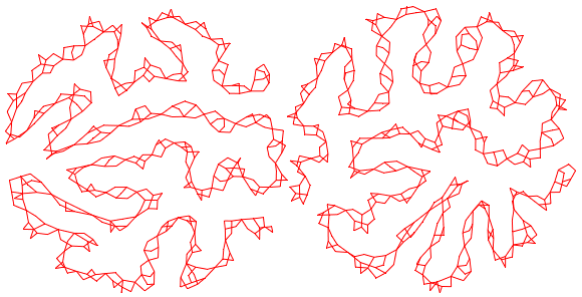

Figure B: The resulted graph of white tailed deer coronvirus genome by Tulip visualization tool (Force Direct layout).