Diffhash

Angel Abdiel Sanquiche-Sanchez

May 4, 2019

1 Abstract

Here we propose an innovative way to find deferentially expressed genes straight from sequencer output. Since assembling can be a very computationally expensive process when not working with arbitrary data, wouldn't it be convenient to assemble only the information that what want and leave behind the useless data. Using a bit of hashing, counting and analyzing we can achieve this.

2 Introduction

The field of bioinformatics faces significant problems with the size, quality and structure of the data extracted from the tissue of organisms of interest. Shotgun sequencing is a method that's used to reduce the time needed to sequence the entire sample of an organism. This usually requires that the data be assembled before anything is done. This is due to the very nature of of shotgun sequencing. Instead of reading the entire sequence from start to end, we shred the sequence into fragments and read as many fragments as we can, until we reach an amount where we can statistically say that we have enough coverage of the sequence to meet the requirements of the experiment were conducting.

Afterwards we would start our traditional pipeline, which in the case is an adaptation of the eel-pond for mRNAseq[1] we call the escambron protocol[2] both implement the khmer protocols[5]. The protocol is setup in various steps. Quality assesment with fastqc[3], trimming with trimmomatic[4], normalizing with normalize-by-median from khmer[7], interleaving with interleave-reads also from khmer[7], assembling using trinityrnaseq[9], transrate[12] to rate the input reads, salmon[11] to quantify and finally extracting expressed genes with edgeR[13]. This is would be a standard pipeline in a differential expresion analisys experiment.

3 Methodology

Our new pipeline differential hash expression or "Diffhash" will change some of this work flow. The initial idea is that assuming a proper random sample of

shotgun sequencing data will cover the entire sequence evenly. Now for a pair of sequences A and B, our null hypothesis is that the distribution of a particular kmer[6] in A is similar to its distribution in B. If the kmers distribution differs significantly it can be considered deferentially expressed if not it can be ignored. With this we can avoid assembling the full sequence and only do a targeted assembly of the regions of interest.

In practice we are using Julia a high-level general-purpose dynamic programming language designed for high-performance numerical analysis and computational science. It version 1.0 released not very long ago that's as good a reason as any to start using it. We will be using the BioSequences package to work with the data and edgeR[13] to do the analysis and Blast to do blast things. Finally there are some shell and python scripts to do some of the grunt work of interleaving and finding matching blast results.

To do a proper comparison of both pipelines we run the standard pipeline until completion to generate known good data. Next we copy the data just after it went through the trimming stage with trimmomatic[4]. This is to remove any bad data that the sequencer knowingly misread and placed "N" noting its placement but not knowing which base corresponds (N can represent any of A,C,T,G).

Now we start Diffhash by generating an empty dictionary. We then go read through every file of sequence A and extract every kmer, count how many times it appears in each file of sequence A and do the same with sequence B. This will generate a hashcount file we pass to our analysis with edgeR[13] which generates a deferentially expressed kmer file. This is where we can now get creative. We can take this deferentially expressed kmer file and turn it into a fasta file by assigning each kmer a header in the form of the kmers line number in the file. Now we can assemble this file with Abyss[10]. This is similar to what Kevlar[8] does. Alternatively we can do a more traditional approach where we take the kmer file and once again go through sequence A and B files. This time we will extract all reads that all the kmers in that read are deferentially expressed. We then use these new filtered list to assemble using trinity[9]. Both of these methods are less computationally complex that to assemble the entire sequence and then do the differential expression analysis. Both have been done in this experiment.

4 Results

Once we have our assembled sequences we need to compare them with the known good sequence. To do this we generated a blast database using ncbi blast plus on the three sequences. Then we blast both sequences to the known good sequence database, we do the reverse blasting our known good against both sequences and the both sequences to each other just to verify how similar they are to each other. This will generate 2 data frames per two comparisons for a total of 6. With the first column of these data frames being the label in the first sequence and its best match in the other sequence. To finish up we go to extract the

first and second column of both of those files flip one or the other to match the sequences and find the intersection and we get these:

	long reads	normal reads
0	TRINITY_DN1_c0_g1_i1	TRINITY_DN21_c0_g3_i1
1	TRINITY_DN0_c0_g1_i1	TRINITY_DN17_c0_g1_i1
2	TRINITY_DN0_c0_g1_i1	TRINITY_DN17_c0_g1_i1
3	TRINITY_DN0_c0_g1_i1	TRINITY_DN17_c0_g1_i1
4	TRINITY_DN0_c0_g1_i1	TRINITY_DN17_c0_g1_i1
5	TRINITY_DN13_c0_g1_i1	TRINITY_DN19_c0_g1_i1
6	TRINITY_DN18_c0_g1_i1	TRINITY_DN16_c0_g1_i21
7	TRINITY_DN11_c0_g1_i1	TRINITY_DN22_c0_g2_i2
8	TRINITY_DN5_c0_g1_i1	TRINITY_DN22_c0_g1_i1
9	TRINITY_DN6_c0_g1_i1	TRINITY_DN23_c0_g1_i1
10	TRINITY_DN15_c0_g1_i1	TRINITY_DN21_c0_g2_i4
11	TRINITY_DN3_c1_g1_i1	TRINITY_DN14_c0_g1_i2
12	TRINITY_DN4_c0_g1_i1	TRINITY_DN21_c0_g1_i7
13	TRINITY_DN14_c0_g1_i1	TRINITY_DN20_c0_g1_i2

	short reads	long reads
0	4	TRINITY_DN13_c0_g1_i1
1	135	TRINITY_DN14_c0_g1_i1
2	301	TRINITY_DN1_c0_g1_i1
3	699	TRINITY_DN11_c0_g1_i1
4	816	TRINITY_DN10_c0_g1_i3
5	959	TRINITY_DN15_c0_g1_i1
6	2270	TRINITY_DN5_c0_g1_i1
7	2982	TRINITY_DN3_c1_g1_i1
8	3438	TRINITY_DN18_c0_g1_i1
9	3747	TRINITY_DN4_c0_g1_i1
10	3907	TRINITY_DN16_c0_g1_i1
11	3914	TRINITY_DN12_c0_g1_i1
12	3924	TRINITY_DN6_c0_g1_i1
13	4290	TRINITY_DN0_c0_g1_i1
14	4745	TRINITY_DN2_c0_g1_i1

	normal reads	short reads
0	TRINITY_DN23_c0_g1_i1	1907
1	TRINITY_DN23_c0_g1_i2	2097
2	TRINITY_DN23_c0_g2_i1	3924
3	TRINITY_DN13_c0_g1_i2	2712
4	TRINITY_DN19_c0_g1_i1	3907
5	TRINITY_DN27_c0_g1_i1	3111
6	TRINITY_DN21_c0_g1_i6	3747
7	TRINITY_DN21_c0_g3_i1	301
8	TRINITY_DN21_c0_g2_i4	959
9	TRINITY_DN12_c0_g1_i1	500
10	TRINITY_DN18_c0_g1_i4	3848
11	TRINITY_DN8_c0_g1_i1	105
12	TRINITY_DN11_c0_g1_i1	3688
13	TRINITY_DN2_c0_g1_i1	410
14	TRINITY_DN16_c0_g1_i21	3438
15	TRINITY_DN15_c0_g1_i1	4812
16	TRINITY_DN17_c0_g1_i3	4290
17	TRINITY_DN17_c0_g1_i4	159
18	TRINITY_DN17_c0_g2_i1	1008
19	TRINITY_DN10_c0_g1_i1	2195
20	TRINITY_DN20_c0_g1_i2	135
21	TRINITY_DN20_c0_g1_i3	3295
22	TRINITY_DN20_c0_g1_i5	2896
23	TRINITY_DN4_c0_g1_i2	2719
24	TRINITY_DN14_c0_g3_i1	5130
25	TRINITY_DN14_c0_g1_i2	2982
26	TRINITY_DN14_c0_g2_i2	3022
27	TRINITY_DN22_c0_g1_i1	2270
28	TRINITY_DN22_c0_g2_i2	5105
29	TRINITY_DN22_c0_g2_i3	699

5 Conclusion

Both of the direct kmer sequence and the filtered sequence generated reciprocal hits in the known good sequence. This is to be expected since were assembling only what we expect to be deferentially expressed in the sequence. And the very fact that these tables are not empty means that there is something to this method. To give even more weight to this concept Kevlar[8] does variant discovery genomes in a similar fashion using differential expression analysis without assembling the sequence first. Additionally the direct and the filtered sequences matched to each other more thoroughly than how they matched to the known good sequence. This is also to be expected since they conceptually should be the same sequence.

6 Acknowledgements

Humberto Ortiz-Zuazaga

- Benevolent dictator for life of Megaprobe.
- Primary investigator.
- Fount of knowledge I've been able to tap.

Louis F. Gil Acevedo

- Much more knowledgeable than me in the biological aspect.
- Stressed Diffhash and identified a significant breaking point.

The aether

- For being the source of the concept.

The rest of the Megaprobe lab team

- At times permit ed me to get my mind to reset and re tackle problems with clarity.
- Made me have to give a good example as one of the seniors in the lab.

7 Future Work

Diffhash is at a very early stage in development. At this point I can easily identify several improvements or features that could be added and explored.

- Parallelizing operations at the read level. Diffhash spends a significant amount of time hashing and not io blocked.
- Utilizing a concept like Count–min sketch in the dictionary.
- Determining a more robust analysis, preferably one more focused on kmer distribution.
- Trying assembling with kmers instead of full reads like Kevlar. One thing to note, there's a limit to how small read size software like trinity is willing to work with.

References

[1] eel-pond for mrnaseq. https://eel-pond.readthedocs.io/en/latest/.

[2] escambron protocols. https://escambron-protocols.readthedocs.io/en/latest/.

[3] S. Andrews. Fastqc. a quality control tool for high throughput sequence data, 2010.

[4] Anthony M. Bolger, Bjoern Usadel, and Marc Lohse. Trimmomatic: a flexible trimmer for Illumina sequence data. *Bioinformatics*, 30(15):2114–2120, 04 2014.

[5] C. Titus Brown, Camille Scott, Michael R. Crusoe, Leigh Sheneman, Josh Rosenthal, and Adina Howe. khmer-protocols 0.8.4 documentation. 12 2013.

[6] Phillip E. C. Compeau, Pavel A. Pevzner, and Glenn Tesler. How to apply de bruijn graphs to genome assembly. *Nat Biotechnol*, 29(11):987–991, Nov 2011. 22068540[pmid].

[7] Michael R. Crusoe, Hussien F. Alameldin, Sherine Awad, Elmar Bucher, Adam Caldwell, Reed Cartwright, Amanda Charbonneau, Bede Constantinides, Greg Edvenson, Scott Fay, Jacob Fenton, Thomas Fenzl, Jordan Fish, Leonor Garcia-Gutierrez, Phillip Garland, Jonathan Gluck, IvÃ¡n GonzÃ¡lez, Sarah Guermond, Jiarong Guo, Aditi Gupta, Joshua R. Herr, Adina Howe, Alex Hyer, Andreas HÃrpfer, Luiz Irber, Rhys Kidd, David Lin, Justin Lippi, Tamer Mansour, Pamela McA'Nulty, Eric McDonald, Jessica Mizzi, Kevin D. Murray, Joshua R. Nahum, Kaben Nanlohy, Alexander Johan Nederbragt, Humberto Ortiz-Zuazaga, Jeramia Ory, Jason Pell, Charles Pepe-Ranney, Zachary N Russ, Erich Schwarz, Camille Scott, Josiah Seaman, Scott Sievert, Jared Simpson, Connor T. Skennerton, James Spencer, Ramakrishnan Srinivasan, Daniel Standage, James A. Stapleton, Joe Stein, Susan R Steinman, Benjamin Taylor, Will Trimble, Heather L. Wiencko, Michael Wright, Brian Wyss, Qingpeng Zhang, en zyme, and C. Titus Brown. The khmer software package: enabling efficient nucleotide sequence analysis. 08 2015.

[8] Fereydoun Hormozdiari Daniel S. Standage, C. Titus Brown.

[9] Manfred G. Grabherr, Brian J. Haas, Moran Yassour, Joshua Z. Levin, Dawn A. Thompson, Ido Amit, Xian Adiconis, Lin Fan, Raktima Raychowdhury, Qiandong Zeng, Zehua Chen, Evan Mauceli, Nir Hacohen, Andreas Gnirke, Nicholas Rhind, Federica di Palma, Bruce W. Birren, Chad Nusbaum, Kerstin Lindblad-Toh, Nir Friedman, and Aviv Regev. Full-length transcriptome assembly from rna-seq data without a reference genome. *Nature Biotechnology*, 29:644 EP –, May 2011.

[10] Shaun D. Jackman, Benjamin P. Vandervalk, Hamid Mohamadi, Justin Chu, Sarah Yeo, S. Austin Hammond, Golnaz Jahesh, Hamza Khan, Lauren Coombe, Rene L. Warren, and et al. Abyss 2.0: resource-efficient assembly of large genomes using a bloom filter. *Genome Research*, 27(5):768–777, Feb 2017.

[11] Rob Patro, Geet Duggal, Michael I Love, Rafael A Irizarry, and Carl Kingsford. Salmon provides fast and bias-aware quantification of transcript expression. *Nature Methods*, March 2017.

[12] Smith-Unna R., C. Boursnell, R. Patro, J. M. Hibberd, and S. Kelly. Transrate: Reference-free quality assessment of de novo transcriptome assemblies. 26(8):1134–1144, 2016.

[13] Mark D. Robinson, Davis J. McCarthy, and Gordon K. Smyth. edger: a bioconductor package for differential expression analysis of digital gene expression data. *Bioinformatics*, 26(1):139–140, Jan 2010. 19910308[pmid].