*Bioinformatics*, 2017, 0–0

doi: 10.1093/bioinformatics/xxxxx[where to look it up in the library]

Advance Access Publication Date: 14 December 2017

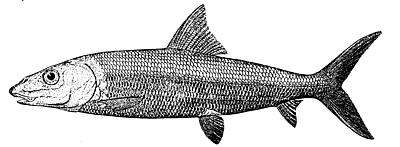
Genome Analysis

|  |
| --- |
| Subject Section[?]  Whole Genome Assembly of Bonefish (Albula glossodonta and A. vulpes)  Austin Alstrom2,\* , Mark Hall1,\*, Jacob Huffaker1,\* and Kaylee Jones1  1Department of Computer Science, Brigham Young University 3361 TMCB PO Box 26576 Provo, UT 84602-6576, 2Department of [Austin and others who aren’t CS add your major and address], Address XXXX etc.  \*To whom correspondence should be addressed.  Associate Editor: Russell Moser  Received on December 14, 2017; revised on \_\_\_; accepted on \_\_\_  Abstract  **Motivation:** Bonefish are an economically important, and threatened group of related species, which can be found in oceans around the world. Unfortunately, the relationships between these species is not well defined. Two bonefish genomes (one from the Pacific Ocean near Hawaii and one from the Atlantic Ocean near Florida) have been sequenced, but neither have been assembled or annotated.  **Results:** We characterized the nuclear genome of the bonefish using next generation sequencing (Illumina [what else was used?]), based on materials collected. The data set consisted of \_\_\_\_\_ reads, which yielded a draft assembly of \_\_\_ contigs and \_\_\_ scaffolds. The estimated genome size was \_\_\_ Gb. \_\_\_ was closely related to the bonefish. This fish contained \_\_\_\_ [protiens and sequences that were the same of something else].  **Availability:**  **Contact:** perry.ridge@byu.edu  **Supplementary information:** Supplementary data are available at *Bioinformatics* online. |

# Introduction

Bonefish are an economically important, and threatened group of related species, which can be found in oceans around the world. “Habitat loss and degredation, as well as illegal harvest” have deteriorated bonefish populations (Bonefish and Tarpon). This species of fish has had a massive commercial and environmental impact in tropical waters, but conservation efforts are limited by a lack of good understanding of the species’ genetic diversity. Bonefish tissue samples have been collected.

The purpose of this project is to sequence their DNA to build a reliable reference genome that will advance research and understanding of this important species. Unfortunately, the relationships between these species is not well defined. Two bonefish genomes, Albula glossodonta and A. vulpes (one from the Pacific Ocean near Hawaii and one from the Atlantic Ocean near Florida), have been sequenced, but neither have been assembled or annotated.



**Figure 1.1 Albula vulpes** A black and white picture of a Bonefish (H. Bigelow et al.)

# Methods

**Genome project history**

We were presented with the lab results of a reading of bonefish DNA, and our endeavor has been to map and assemble these reads into the original genome. Extrapolating from the size and amount of reads in one of the smaller files, we have estimated that about 700 million reads were present in the information given to us. These were offered in the form of .fastq files, with paired reads listed within the same files for each file.

**Genome sequencing and assembly**

For implementing our code for this problem, we selected the Python programming language for its efficiency and presence of useful libraries to help us with our solution. We outlined a process for assembling the genome, including preprocessing the reads, assembling these processed reads into contigs, and assessing the contigs for accuracy. Considering the heavy computational demand of this project, we have used the BYU Mary Lou Fulton Supercomputer to run this code.

For our preprocessing, we noted that each of the reads in the .fastq files are accompanied by a string of ASCII characters representing the confidence of the reads. Assuming an excess amount of reads for each locus in the genome, we opted to pare down the original set of reads by eliminating reads that failed to meet a certain level of confidence. The Biopython library offered a simple way to do this, of which we availed ourselves. We assessed each read by Biopython’s ‘phred\_quality’; experimenting with one of the smaller files, we found that a confidence level of 10 eliminated about half of the reads while preserving the (admittedly trivial) contigs that our code generated from those data, so we elected to use that confidence baseline for our implementation.

In order to further simplify the problem, we assume that any accurate reads should still be redundantly present in the files given, and so we are experimenting with a cardinality approach to further attempt to eliminate inaccurate reads. We assume that a good read should still be present in at least two or three instances in these data. The Python ‘ntCard’ library provides an efficient method to confirm that reads are appropriately numerous, and we intend to use this in our next run of this code.

After preprocessing, we use a De Bruijn graph approach to assembling the genome. In light of the sheer scale of the problem, we came to the conclusion that our data set was too large to be held in RAM, and so we opted for a generative approach. Keeping in mind efficiency concerns for data access and storage, we chose to use a SQLite database to represent the De Bruijn graph. Using the k-1 length prefixes as the primary key for the database yields worst-case O(log n) look-up time for these prefixes, and so we implemented the database accordingly, with each row consisting of a prefix, a marker to label visitation in the Eulerian graph algorithm, and the list of all suffixes (theoretically up to 4) associated with each prefix. We generated this database using each of the reads that resulted from the confidence baseline preprocessing.

Our initial approach to compiling the genome from this database has been a brute-force algorithm. We intend to use a more efficient Eulerian path implementation, but we intend to confirm the functionality of the previous steps before completely implementing this algorithm. The brute-force algorithm iterates through each prefix in the graph, and creates a path as long as possible starting with each prefix, mapping subsequent suffixes to prefixes until it comes to a suffix absent in the graph. Contigs of significant length that are formed this way are published to a text file.

For post-processing, we intend to use the original pairing of the reads to confirm that reads are approximately an expected distance away from each other within the contig. Our research indicates that distances of approximately 500 base pairs between reads are common between paired reads, so we intend to assess sufficiently large contigs to ensure that ostensibly paired reads meet reasonable expectations like these.

As necessary, we have considered performing shotgun analysis on the formed contigs to finish assembling the genome. We don’t know how exact or how long the contigs formed by the program will be, but we will attempt to write code to handle this as appropriate. Once this is complete, we expect to have something like the entire genome of the bonefish assembled.

We could not assume that pairs read in were exact distances apart, so we ended up treating them as individual reads, knowing that the latter in the pair came later in the sequence. The paring was helpful in analyzing “bubbles,” or contingencies in the reads, in the graph to confirm the contig at the end.

While running the first version of the algorithm, it found 7 contigs, 6 of length 152 (with 2 consecutive reads), and 1 of length 153 (with 3 consecutive reads). There were 23 instances where the given reads differed only by the last base pair, so 23 ambiguous prefix to suffix mappings were completed. These were mapped to, or referenced, 5 separate times in our brute-force approach in the implementation, which would result in more contigs. However, with this first implementation, we were ignoring a lot of necessary details, so we decided to not pay attention to the said prefix and suffixes at the time.

With this first run, we could not thing of the means in which to dynamically sort text files, so we ended up suing database-style files and SQLite. The efficiency of doing so matches that of a binary search on a sorted text file. With the two (relatively) small fastq file inputs, the database files ended up being roughly 100MB; equal to 140% of the memory used of the input fastq files. It ran in 14.37 seconds. This could potentially work with a BioPython module that can output to SQL.

This implementation was unfortunately too long, but let to us working on threading the program, and reconsidering the requested memory needed on the supercomputer (Mary Lou) running the algorithm. The pre-processing for quality reads took roughly 10 to 15 hours, and making the subsequent database would take longer.

**Genome annotation**

[pending project completion]

# Results

[pending completion of the program]

We characterized the nuclear genome of the bonefish using next generation sequencing (Illumina [what else was used?]), based on materials collected. The data set consisted of \_\_\_\_\_ reads, which yielded a draft assembly of \_\_\_ contigs and \_\_\_ scaffolds. The estimated genome size was \_\_\_ Gb. \_\_\_ .

### The Assessment

In experimenting with the implementation described above, we have discovered several potential improvements that we intend to use in future uses of our method. Our implementation is not devoid of fallibility, but we hope to reduce error and inaccuracy as much as possible through experimentation and analysis of results.

Among the most significant results we noted with our initial run of our algorithm was the true extent of the complexity of our project. The time it took to run our code exceeded our expectations. For example, it took between 10 and 15 hours for Biopython to iterate through all files and parse out unwanted reads. Adapting these reads into our SQLite database took longer still, even after we adjusted our request for computational resources with the supercomputer to accommodate for the use of more memory.

One of the solutions we’ve contemplated for this issue is running asynchronous processes using threading or similar concepts. Our original naive implementation was, unfortunately, primarily synchronous; we failed to take full advantage of the computational resources afforded us by the Fulton Supercomputer. In subsequent renditions, we intend to consider ways to make our solution more efficient, such as splitting our implementation into simultaneously executable asynchronous processes, as we find to be plausible.

**In Conclusion**

[pending project completion]

Acknowledgements

The Fulton Supercomputing Lab was a pivotal part of our project.

The initial reads that we were provided with were also crucial to the completion of our project.

Funding

This work has been supported by the Bioinformatis Department of Brigham Young University. [name specific people? Like the professor and the TA?]

*Conflict of Interest:* none declared.

References

Bonefish & Tarpon Trust (2017) Bonefish Research. Bonefish & Tarpon Unlimited (BBT), Coral Gables, Florida.

H.B. Bigelow and W.C. Schroeder (1953) The Fishes of the Gulf of Maine, US Department of Interior, USA.