Scaffolder - Software for Reproducible Genome Scaffolding.

Michael D Barton[1]*, Hazel A Barton[1]

[1] Department of Biological Sciences, Northern Kentucky University, Nunn Drive, Highland Heights, KY 41076

Email: Michael D Barton*- mail@michaelbarton.me.uk; Hazel A Barton - bartonh@nku.edu;

*Corresponding author

Abstract

Background: Assembly of short-read sequencing data can result in a fragmented non-contiguous series of genomic sequences. Therefore a common step in a genome project is to join neighbouring sequence regions together and fill gaps in the assembly using additional sequences. This scaffolding step, however, is non-trivial and requires manually editing large blocks of nucleotide sequence. Joining these sequences together also hides the source of each region in the final genome sequence. Taken together, these considerations may make reproducing or editing an existing genome build difficult.

Methods: The software outlined here, "Scaffolder," is implemented in the Ruby programming language and can be installed via the RubyGems software management system. Genome scaffolds are defined using YAML - a data format, which is both human and machine-readable. Command line binaries and extensive documentation are available.

Results: This software allows a genome build to be defined in terms of the constituent sequences using a relatively simple syntax to define the scaffold. This syntax further allows unknown regions to be defined, and add additional sequences to fill gaps in the scaffold. Defining the genome construction in a file makes the scaffolding process reproducible and easier to edit compared with FASTA nucleotide sequence.

Conclusions: Scaffolder is easy-to-use genome scaffolding software. This tool promotes reproducibility and continuous development in a genome project. Scaffolder can be found at http://next.gs.

Background

High-throughput sequencing can produce hundreds of thousands to millions of sequence reads from a genome. At the time of writing, high-throughput sequencing is limited to producing reads less than 1,000 nucleotides in length. Therefore to resolve a sequence longer than this, such as a complete genome, these numerous smaller fragments must be pieced together. The process of joining reads into longer sequences is the 'assembly' stage of a genome project [1].

Assembly software takes the nucleotide reads produced by sequencing hardware and, in the ideal case, outputs a single complete genome sequence composed of these individual fragments. An analogy for this process is a jigsaw puzzle: each nucleotide read represents a single piece, and the final genome sequence is the completed puzzle. Sequences of repetitive nucleotide 'repeat' regions within the genome, or biased and incomplete sequencing data, may prevent the genome being assembled into a continuous sequence. This may be due to not enough, or multiple different overlaps between reads and is analogous to missing pieces in the jigsaw or pieces that fit to multiple other pieces.

The advent of high throughput sequencing methods has led to a renewed interest in algorithms to solve the problem of genome assembly [2,3]. Nonetheless, due to the complexity of merging large numbers of overlapping reads genome assembly software may be unable to produce a complete sequence. Instead, the algorithm may generate several large assembled regions of sequence ('contigs') composed of the many individual reads that represent assembly into the longest possible sequences, given the data. Nonetheless, these contigs represent a fragmented picture of the genome and require additional work to join together into a complete sequence.

The process of ameliorating a draft sequence into a finished continuous sequence can be expensive in terms of time and laboratory effort, although the genomic data present in a set of generated contigs may be sufficient for many research questions [4]. Nevertheless, a continuous high-quality 'finished' genome sequence does provide a greater depth of information, such as complete resolution of repeat regions and precise estimates of distances between genomic elements [5,6]. The process of joining these contigs together to form a continuous genome sequence is called the 'scaffolding' or 'finishing' stage and is the focus of the software described in this article.

Scaffolding

Scaffolding is the process of joining a series of disconnected contigs into a complete continuous genome sequence. Due to genomic complexity and missing data, scaffolding may not ultimately produce a final completed sequence, but may still succeed in joining a subset of contigs together or resolving gaps between contigs. An overview of the required steps in the scaffolding process is outlined below:

Contig Orientation

The sequencing process generates reads from either strand of the DNA helix and the resulting contigs constructed from these reads may represent either the 5' $\rightarrow$ 3' or 3' $\rightarrow$ 5' DNA strands with respect to the origin of replication. Orientating all contigs to point in the same direction requires reverse complementing sequences where necessary. In the case of microbial genomes this orientation will be to the 5' $\rightarrow$ 3' direction following the direction of genome replication.

Contig Ordering

Contig ordering determines the placement of observed contigs to best represent their order in the true genome sequence. The correct placement of each contig also highlights any extra-genomic DNA, such as plasmids which are scaffolded separately from the genomic sequence. The order is commonly started at the contig containing the origin of replication. All subsequent contigs are then ordered in the 5' $\rightarrow$ 3' direction of DNA replication.

Contig Distancing

Given the correct order and orientation, determining the distance between contigs results in an estimate of the complete genome size. The size of any inter-contig gaps represents the length of an unknown region in the genome. Filling these regions with unknown nucleotide characters ('N') allows a draft continuous sequence, which is useful for representing both the known and to-be-resolved areas in the genome sequence.

Gap Closing

During the scaffolding process, closing and filling gaps between contigs completes and improves the genome scaffold. Closing gaps may require returning to the laboratory to perform additional sequencing or using computational methods of estimating the unknown sequence. This additional sequence is then used to replace the gap between two contigs, joining them into a single sequence. Once all contigs have been joined and gaps in a scaffold closed, the genome may be considered finished.

Computational Methods for Scaffolding

The process of resolving a genome sequence and building a scaffold uses wet laboratory methods, *in silico* methods, or a combination of both. An example of a computational method might use the existing paired-read data from the sequencing stage where reads were produced in pairs at a known distance apart in the genome. The occurrence of paired reads in separate contigs can be used to probabilistically estimate the order and distance between these contigs. Alternatively, laboratory methods may use PCR to amplify the unknown DNA in a gap region, then use traditional Sanger sequencing to determine the sequence of this gap. Computational methods are more preferable as they are less costly in laboratory time and materials compared to manual gap resolution, and use the available data that exists following sequencing [7]. Finally when the scaffold cannot be completely resolved, *in silico* software packages exist to suggest the likely primers necessary for PCR amplifying the sequence in gap regions [8].

Examples of *in silico* methods include using synteny to compare the assembled contigs to a complete reference genome sequence by searching for areas of sequence similarity between the two. Any areas of corresponding sequence with the reference genome can be used to infer contig placement and build the contigs into a scaffold [9–11]. Recombination within each of the two compared genomes can, however, reduce the effectiveness of this analysis.

Repeat regions may also be responsible for multiple gaps when building a genome sequence; tandemly repeated nucleotide regions in the genome produce multiple reads with similar sequence. As many assembly algorithms rely on sequence overlaps between reads to build a contig, the similarity between repeat-region reads can result in the assembly collapsing into an artificially short sequence or being ignored by more conservative assembly algorithms. Such regions can be resolved by using algorithms specifically reassemble the collapsed repeat region correctly [12, 13], while a related approach uses unassembled sequence reads matching the regions around a scaffold gap to construct a uniquely overlapping set of reads across it [14].

Paired-read data can provide an extra level of information about how contigs may be scaffolded together. Heuristic scaffolding algorithms take advantage of this data to search for the optimal configuration of contigs in the scaffold that matches these paired-read distances [15, 16]. Synteny data from a reference genome can also be combined with this paired-read data to estimate the best contig configuration [17].

These described *in silico* methods provide a wide array of approaches for merging contigs into a larger, continuous scaffold sequence. Nevertheless the scaffolding process may still require manually inserting additional sequences or further joining contigs using PCR-derived sequence. Moving and editing large blocks of nucleotide text by hand however possibly introduces human error and precludes any reproducibility of the steps in this process.

The software outlined here, "Scaffolder," aims to address these problems of reproducibility by creating a file syntax and software framework for editing a genome scaffold. Scaffolder uses a specific file format to define how contigs are joined, additional sequences are inserted, and for the specification of unknown regions. This syntax allows a scaffold to be updated by simply editing the scaffold file. As such, scaffolder facilitates a reproducible scaffolding process and provides a concise overview of how the final genomic scaffold was constructed.

Implementation

Code and Dependencies

Scaffolder is written in the Ruby programming language using version 1.8.7 [18]. The scaffolder package is split into two libraries. The first called "scaffolder" provides the core Scaffolder application programming interface (API). The second library "scaffolder-tools" provides the Scaffolder command line interface (CLI).

Unit tests were implemented to maintain individual elements of the source code during development and were written using the Shoulda and RSpec [19] libraries. Integration tests were written to test the Scaffolder software interface as a whole and were written using the Cucumber library [19].

The Scaffolder source code is documented using the Yard library [20]. Unix manual pages for the command line were generated using the Ronn library [21]. The manipulation of biological sequences in Scaffolder uses the BioRuby library [22]. A full list of the software dependencies in Scaffolder can be found in the Gemfile in the root of each source code directory.

Scaffold File Syntax

The choice of nucleotide sequences comprising the scaffold is specified using the YAML syntax [23]. YAML is a language format using whitespace and indentation to produce a machine readable structure. As YAML is a standardised data format, third-party developers have the option to generate a genome scaffold using any programming language for which a YAML library exists. The YAML website lists current parsers for languages including C/C++, Ruby, Python, Java, Perl, C#/.NET, PHP, and JavaScript. In addition to being widely supported, YAML-formatted scaffold files can be validated for correct syntax using third-party tools such as Kwalify [24].

Initial sequencing data assembly may result in an incomplete genome build. Generation of additional sequences either through PCR or computational methods means that genome scaffolding may be an on-going process. The scaffold file should therefore be simple to update manually in addition to being computationally tractable. This requirement was also best suited to YAML syntax which is human-readable and simple to edit in a standard text editor.

The scaffold file takes the form of a list of entries. Each entry corresponds to a region of sequence used in the final scaffold sequence. Each entry in the scaffold file may have attributes that define whether a sub-sequence or the reverse complement of the sequence should be used. The types of attributes available, and an example scaffold file are outlined in the Results section.

Input sequence data and scaffolding process

The input data for Scaffolder is nucleotide sequence in FASTA format file. These nucleotide sequences can be of any length and for example may be individual sequences, assembled contigs or contigs which have been joined to into larger scaffolds. The case in which Scaffolder may be most useful is using the contigs and scaffolded contigs produced from the previously assembled set of sequencing reads, combined with additional gap filling sequences produced by PCR or *in silico* methods as outlined in the Background.

The scaffolder algorithm reads through the scaffolder file parsing. Each entry is then converted to corresponding nucleotide sequence by fetching from the FASTA file and then performing any designated sequence trimming or reverse complementing. Each of the sequence entries are then joined into a continuous single super-sequence and returned to the console output.

Results

Scaffolder Simplifies Genome Finishing

The Scaffolder software facilitates reproducibly joining nucleotide sequences together into a single contiguous scaffolded super-sequence. Plain-text scaffold files written in the YAML syntax specify how these sequences should be joined and the scaffolder software is used to generate the scaffold sequence from these instructions. In addition to specifying which contigs are required, the scaffold file allows the contigs to be edited into smaller sub-sequences or reverse complemented if necessary. The process of genome finishing may involve producing additional oligonucleotide sequences to fill unknown regions in the scaffold and the Scaffolder allows these additional insert sequences to be used to fill these gaps. These inserts can also be treated in the same manner as larger contig sequences: trimmed and/or reverse complemented to match the corresponding gap region size and orientation.

The distances between contigs may be estimated from paired-read data or from mapping the contigs to a reference genome. These inter-contig gap regions are useful to join separate sequences together by an estimated size and the scaffold file allows for the specification of such unresolved regions by inserting regions of 'N' nucleotides into the scaffold. The use of these regions in the scaffold indicates the unresolved regions in the build and their approximate size.

The scaffold file specifies how sequence regions are joined together; however, the sequences themselves are not stored in the scaffold file and are instead maintained as a separate FASTA file. Nucleotide sequences in a FASTA-format file are the standard output for a genome assembler, and these may be contigs, sets of contigs scaffolded into larger sequences, or both. These nucleotide sequences stored in the FASTA file are referenced in the scaffold using the first word from in the FASTA header of the corresponding sequence. Maintaining the nucleotide sequences in a separate file preserves the unedited sequence and decouples the data from the specification of how it should be used to produce the genome sequence.

Defining a Scaffold as a Text File

The scaffold file is written using the YAML syntax and an example is shown in Figure 1. This file illustrates the text attributes used to describe a scaffold and how the sequences are correspondingly joined together in the genome build. The basic layout of the scaffold file is a list of entries, where each entry corresponds to a region of sequence in the generated scaffold super-sequence.

Simple sequence region

The first line of the scaffold file in Figure 1 begins with three dashes to indicate the start of a YAML-formatted document. The first entry (highlighted in green) begins with a dash character '-' to denote an entry in the YAML list. This is a requirement of the YAML format: that each entry begins with a dash line. The next line is indented by two spaces where whitespace is required to group similar attributes together. The "sequence" tag indicates that this entry corresponds to a sequence in the FASTA file and the following line indicates the name of this sequence using the "source" tag. The first word of the FASTA header is used to identify which sequence is selected from the file. Together these three lines describe the first entry in the scaffold as a simple sequence using the sequence named 'sequence1'. On the right hand side of Figure 1 this produces the first region in the scaffold, also shown in green.

Unresolved sequence region

The second entry in the scaffold, highlighted in orange, is identified by the "unresolved" tag, indicating a region of unknown sequence but known length. The second line specifies the size of this unknown region. This entry produces a region of 20 'N' characters in the scaffold.

Trimmed sequence region with multiple inserts

The last entry in the scaffold, highlighted in blue, adds the sequence named 'sequence2' to the scaffold. This entry demonstrates how this sequence may be manipulated prior to addition to the scaffold. The 'start' and 'stop' tags trim the sequence to these coordinates inclusively, while the "reverse" tag instructs scaffolder to also reverse complement the sequence. In the putative scaffold shown in Figure 1 this completes the scaffold sequence.

The final entry in the scaffold also includes the "inserts" tag to add additional regions of sequence. These inserts are also added as a YAML list, with each insert beginning with a dash. The first insert, shown in purple, uses similar attributes to that of a sequence entry; the reverse, start and stop tags are used to trim and reverse complement the insert. Similarly the 'source' tag identifies the corresponding FASTA sequence as 'insert1'. The "open" and "close" tags are specific to inserts and determine where the insert is added in the enclosing sequence. The region of the sequence inside these coordinates is inclusively replaced by the specified insert sequence. This is visualised in the putative scaffold in Figure 1 by the black lines bisecting the blue sequence.

The next insert, shown in brown, is specified using only the ‘open’ tag. This illustrates that only one of either ‘open’ or ‘close’ tags is required when adding an insert sequence. If only one of the ‘open’ or ‘close’ tags is used the corresponding opposing ‘open’/‘close’ coordinate is calculated from the length of the insert FASTA sequence. This allows inserts to bridge into, and partially fill gap regions, without requiring an end coordinate position.

Scaffolder Software Interface

Scaffolder provides a standardised set of Ruby classes and methods (API) for interacting with the scaffold. This allows Scaffolder to be integrated into existing genomics workflows or used with Ruby build tools such as Rake. In addition Scaffolder provides a command line interface (CLI) to validate the scaffold file and build the draft super sequence. The Scaffolder CLI behaves as a standard Unix tool and returns appropriate exit codes and manual pages. The use of both these Scaffolder interfaces is outlined in detail on the scaffolder website (http://next.gs).

Discussion

Scaffolding an incomplete genome assembly requires joining contigs and additional gap-filling sequence using a combination of computational and laboratory methods. The process of manually editing a scaffold is inherently hard to reproduce and introduces the possibility of irreducible techniques and/or human error. In respect to these drawbacks the aims of the Scaffolder software are twofold: 1) to provide software that is easy to install and simplifies the task of genome finishing; and 2) to facilitate reproducibility in the scaffolding and finishing stage of a genome project.

Scaffolder was designed to be as simple to use and, assuming the Ruby and RubyGems software are present, can be installed in a single command line step. This should negate any possible barriers to entry which require manual compilation of source code. Scaffolder also uses a minimal and compact syntax to describe how the genome scaffold sequence should be generated. This syntax is simple to construct and edit whilst being succinct and readable. The goal of this syntax is to make genome scaffolds easy to write in a common text editor.

Constructing a genome using by specifying the scaffold organisation in text file makes generating a scaffold super sequence both reproducible and deterministic for the same file and set of FASTA sequences. In comparison, joining large nucleotide sequences by hand cannot be reliably reproduced. The scaffold file furthermore provides a human readable description of how the scaffold is constructed. Configuring the final sequence in the scaffold file means the build is easier to edit once already constructed. The YAML text format allows comparison of differences between scaffold builds using standard Unix tools such as diff. This makes scaffold files easier to store in version control systems and allows genome finishers to use methods similar to those in software development.

Conclusions

Scaffolder is software aimed at both bioinformaticians and biologists familiar with the command line who wish to build a genome scaffold from a set of contigs. The Scaffolder file format maintains the genome scaffold as a concise and readable text representation that allows third parties to see how the genome sequence was scaffolded. This file format also allows a broad overview of which sequences were included and how they are ordered in the genome scaffold, something not possible to deduce from a megabase-length string of nucleotide characters. Scaffolder furthers increases the ease of reproducibility in genome projects by allowing the scaffold super-sequence to be reliably reproduced from the same scaffold file. The YAML syntax for writing the scaffold file is also standardised and simple to manipulate programmatically. This thereby means the scaffolding process follows the Unix tenet of "If your data structures are good enough, the algorithm to manipulate them should be trivial."

Availability and Requirements

Project name: Scaffolder v0.4.1, Scaffolder Tools v0.1.2

Project home page: http://next.gs

Operating system: Platform Independent. Tested on Mac OS X and Ubuntu.

Programming language: Ruby

Other requirements: RubyGems

License: MIT

Any restrictions to use by non-academics: None

Competing Interests

The authors declare no competing interests.

List of Abbreviations Used

API: Application programming interface

CLI: Command line interface

PCR: Polymerase chain reaction

YAML: YAML ain't markup language [23]

Authors contributions

MDB developed and maintains the Scaffolder tool. MDB and HAB wrote the manuscript.

Acknowledgements

This work was supported by the National Institute for Health: IDeA Network of Biomedical Research Excellence (KY-INBRE) grant (NIH 2P20 RR016481-09) and the NIH R15 AREA Program grant (R15GM079775).

References

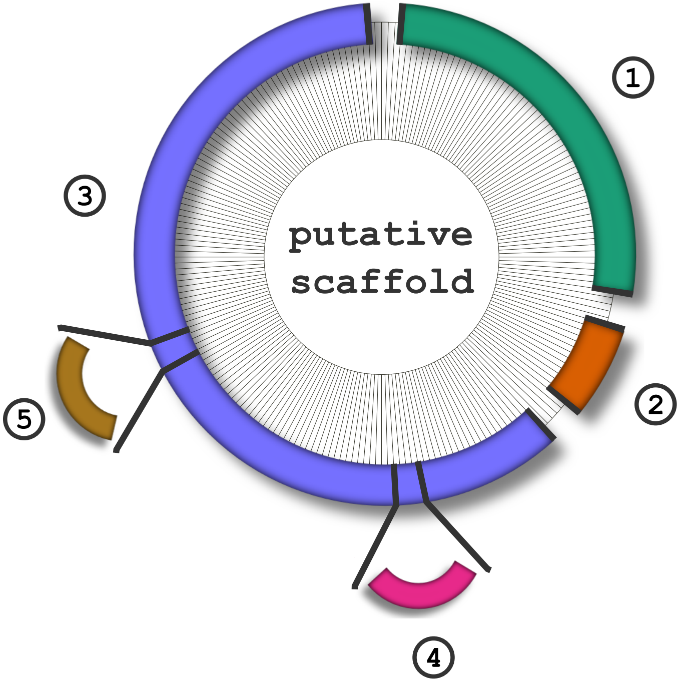
1. Miller JR, Koren S, Sutton G: **Assembly algorithms for next-generation sequencing data**. *Genomics* 2010, **95**(6):315–327.
2. Pop M, Salzberg SL: **Bioinformatics challenges of new sequencing technology.** *Trends in Genetics* 2008, **24**(3):142–149.
3. Pop M: **Genome assembly reborn: recent computational challenges**. *Briefings in Bioinformatics* 2009, **10**(4):354–366.
4. Branscomb E, Predki P: **On the high value of low standards.** *Journal of Bacteriology* 2002, **184**(23):6406–6409.
5. Parkhill J: **The importance of complete genome sequences.** *Trends in Microbiology* 2002, **10**(5).
6. Fraser CM, Eisen JA, Nelson KE, Paulsen IT, Salzberg SL: **The value of complete microbial genome sequencing (you get what you pay for).** *Journal of Bacteriology* 2002, **184**(23).
7. Nagarajan N, Cook C, Di Bonaventura M, Ge H, Richards A, Bishop-Lilly KA, DeSalle R, Read TD, Pop M: **Finishing genomes with limited resources: lessons from an ensemble of microbial genomes.** *BMC Genomics* 2010, **11**:242+.
8. Gordon D, Desmarais C, Green P: **Automated finishing with autofinish.** *Genome Research* 2001, **11**(4):614–625.
9. Richter DC, Schuster SC, Huson DH: **OSLay: optimal syntenic layout of unfinished assemblies**. *Bioinformatics* 2007, **23**(13):1573–1579.
10. Zhao F, Zhao F, Li T, Bryant DA: **A new pheromone trail-based genetic algorithm for comparative genome assembly.** *Nucleic Acids Research* 2008, **36**(10):3455–3462.
11. Assefa S, Keane TM, Otto TD, Newbold C, Berriman M: **ABACAS: algorithm-based automatic contiguation of assembled sequences.** *Bioinformatics (Oxford, England)* 2009, **25**(15):1968–1969.
12. Mulyukov Z, Pevzner PA: **EULER-PCR: finishing experiments for repeat resolution.** *Pacific Symposium on Biocomputing. Pacific Symposium on Biocomputing* 2002, :199–210.
13. Koren S, Miller JR, Walenz BP, Sutton G: **An algorithm for automated closure during assembly.** *BMC Bioinformatics* 2010, **11**:457+.
14. Tsai IJ, Otto TD, Berriman M: **Improving draft assemblies by iterative mapping and assembly of short reads to eliminate gaps.** *Genome Biology* 2010, **11**(4):R41+.
15. Dayarian A, Michael TP, Sengupta AM: **SOPRA: Scaffolding algorithm for paired reads via statistical optimization.** *BMC Bioinformatics* 2010, **11**:345+.
16. Boetzer M, Henkel CV, Jansen HJ, Butler D, Pirovano W: **Scaffolding pre-assembled contigs using SSPACE.** *Bioinformatics* 2011, **27**(4):578–579.
17. Pop M, Kosack DS, Salzberg SL: **Hierarchical scaffolding with Bambus.** *Genome Research* 2004, **14**:149–159.
18. Matsumoto Y: **The Ruby Programming Language**[http://www.ruby-lang.org/].
19. David Chelimsky and Dave Astels and Bryan Helmkamp and Dan North and Zach Dennis and Aslak Hellesoy: *The RSpec Book: Behaviour Driven Development with Rspec, Cucumber, and Friends.* Pragmatic Bookshelf 2010.
20. Segal L: **YARD - A Ruby Documentation Tool**[http://yardoc.org/].
21. Tomayko R: **Ronn - manual page authoring tool**[http://rtomayko.github.com/ronn/].
22. Goto N, Prins P, Nakao M, Bonnal R, Aerts J, Katayama T: **BioRuby: bioinformatics software for the Ruby programming language.** *Bioinformatics (Oxford, England)* 2010, **26**(20):2617–2619.
23. Evans CC: **YAML: a human friendly data serialization standard for all programming languages**[http://www.yaml.org/].
24. Kuwata Lab: **Kwalify: schema validator and data binding for YAML/JSON**[http://www.kuwata-lab.com/kwalify/].

Figures

Figure 1 - Example of Scaffolder File and the Resulting Build

An example scaffold file written using the YAML syntax [23] (left) and the resulting putative scaffold sequence (right). The scaffold contains three entries and two inserts. Each entry in the scaffold file text is delimited by a '-' on a new line and highlighted using separate colours. The scaffold diagram on the right is not to scale and instead illustrates how the scaffold sequences are joined.

```
       ---
       -
①        sequence:
           source: 'sequence1'
       -
②        unresolved:
           length: 20
       -
         sequence:
           source: 'sequence2'
③          start: 30
           stop: 1000
           reverse: true
           inserts:
           -
             source: 'insert1'
             start: 8
             stop: 160
④            reverse: true
             open: 200
             close: 250
           -
⑤            source: 'insert2'
             open: 400
```



putative scaffold
1
2
3
4
5