PhylogeneticGraph
Program Documentation
Version 0.1

Ward C. Wheeler
Division of Invertebrate Zoology,
American Museum of Natural History,
200 Central Park West, New York, NY, 10024, USA;
wheeler@amnh.org

April 8, 2022

Contents

1 Introduction **3**

2 Overview of Code base **3**

3 Character reorganization and optimizing **3**
- 3.1 Character weights . . . 3
- 3.2 BitPacking . . . 3

4 Command Parsing **4**

5 Post-Order Graph Traversal **4**
- 5.1 Trees . . . 4

6 Pre-Order Graph Traversal **4**
- 6.1 Trees . . . 4

7 Adding New Character Types **5**
- 7.1 Execution in Parallel . . . 7

8 Bibliography **8**

1 Introduction

This document contains descriptions of algorithms, procedures, data structures and other aspects of the source code documentation for the program PhylogeneticGraph (`PhyG`).

`PhyG` is a sucessor program to `POY` [2, 7, 4, 5, 8, 9] https://github.com/wardwheeler/POY5, although a "complete" Haskell rewrite, optimized C (and even some assembler) was ported over from POY for pairwise alignment of small alphabet (¡8) sequences. These functions are access via the Haskell FFI.

2 Overview of Code base

Source code structure.

3 Character reorganization and optimizing

Input data are passed through several fuincionts to:

- Rename taxa
- Exclude taxa
- Add missing data for taxa not present in all input files
- Check that input taxa and any input graphs contain teh same leaf set
- Data a reblocked if specified
- Static (Non-additive, additive, and matrix) characters are reorganized so that each class is put in a single (extensive) character (one for each type) type for each block.
- Non-additive characters are 'bit-packed' into new characters with state numbers $=2$, ≤ 4, ≤ 5, ≤ 8, ≤ 64, and > 64. Invariant charcaters are filtered out.

3.1 Character weights

Static characters (Non-additive, additive, and matrix) with integer weights are reorganized by repeating th echaracter the number of times of its weight. This is to avoid alot of unnecessary ($\times 1$) operations. Non-integer weight characters are not reorganized or bit packed.

3.2 BitPacking

Non-additive characters, either from input or static approximation recoding, are initially encoded as little-endian bit-vectors. These are flexible, with unbounded (more or less)state numbers. They are also inefficient in space nad optimization compared with native types such as Word64.

An additional level of inefficiency comes from encoding characters with lower numbers of states in type (e.g. word64) with a single bit for each state. As a result, non-additive characters are recoded according to their number os states into 5 new character types (in Types.hs): Packed2, Packed4, Packed5, Packed8, and Packed64.

Characters with > 64 states remain (although a new character is created) as bit-vectors. Characters with 2, 3-4, 5, and 6-8 states are "packed" into 64 bit Word64 with 32, 16, 12, and 8 characters per word respectively. Characters with 9-64 states are encoded as a single Word64. These are concatenated into a Vector and stored in characterData as "packedNonAddPrelim" and "packedNonAddFinal".

Non-additive charterers are first separated into groups of like-state number, then appropriate ercodign and packing are applied. One important point is that states may be non-sequential (ie 3 and 23), yet still binary (after accounting for "missing" = all bits On). THe are identifiued and recoded as seqeuntial states (bits 0,1,2..).

Simultaneous 2-median and 3-median (post-order and pre-order) optimization is performed via the methods described in [10] and [3] modified for 64-bit words and alternate state numbers (basically mask and shit numbers). These methods, documented and undocumented likely originate with D. E. Knuth ina general sense, and [1] in phylogenetic software.

The module Input.BitPack.hs contains the functions and constants (e.g. masks) to deal with bit-packed characters.

4 Command Parsing

5 Post-Order Graph Traversal

5.1 Trees

A decorated Graph (tree) is created for each character for each block for the graph. For exact characters, where no addition traversals are required, the specified or default outgroup sets the direction of the graph. For non-exact (e.g. sequence) characters the best traversal rooting is stored for each character in each block although the cost of the graph is recalculated based on the best traversal (over all edges in the graph), the preliminary (post-order) states are not propagated back to the decorated graph (third field of phylogenetic graph). After the pre-order pass, the final states are propagated back. Vertices are not renumbered during the rerooting process, so indices remain unchanged.

Preliminary states (post-order) are determined for exact and non-exact characters as in [6].

6 Pre-Order Graph Traversal

6.1 Trees

Final state assignments of root vertices are set to the preliminary, post-order state. Final states are propagated back to the decorated graph (third field of phylogenetic graph). Vertices are not renumbered during the rerooting process, so indices remain unchanged.

Final states (pre-order) are determined for exact and non-exact characters as in [6]. Currently final states for non-exact characters (e.g. sequence) are set as the median between the gapped preliminary state of the vertex and the final state of its parent (for a tree), 'extra' gaps in preliminary state are propagated to the gaped left and right descendant sequences, left, right, and parent final sequences should now line up and a 3-median can be calculated.

7 Adding New Character Types

Current character types include Additive, Non-Additive, Matrix, Slim Sequences, Wide Sequences, and Huge Sequences. Functions that branch on character types need to be updates and are found in:

- GraphOptimization.Medians.hs
  - Median2Single
  - Median2SingleStaticIA
  - Union2Single
  - GetPrealignedUnion
  - getPreAligned2Median
  - median2SingleNonExact
- GraphOptimization.PreOrderFunctions.hs
  - updateCharacter
  - getCharacterDistFinal
  - setFinal
  - setPrelimToFinalCharacterData
- Commands.Transform.hs
  - transformCharacter
- Commands.CommandExecution.hs
  - makeCharLine
  - getCharCodeInfo
  - makeBlockCharacterString
  - pairList2Fasta
  - getCharacterString
- Types.Types.hs
  - CharType
  - nonExactCharacterTypes
  - exactCharacterTypes
  - prealignedCharacterTypes
  - CharacterData
  - emptyCharacter
- Utilities.Utilities.hs

  - getCharacterInsertCost
  - splitBlockCharacters
  - getNumberExactCharacters
  - getNumberSequenceCharactersC9l7mb7s!
  - getCharacterLength
- Utilities.ThreeWayfunctions.hs
  - threeMedianFinal
- Support.Support.hs
  - subSampleStatic
  - makeSampledPairVect
- Input.Reorganize.hs
  - removeConstantChars
  - filterConst
  - getVariableChars
  - getVarVecBits
  - assignNewField
  - organizeBlockData'
- Input.FastAC.hs
  - Functions for sequence data processing on input
- Input.DataTransformation.hs
  - These are for input–so not used by static approx or bit-packed
  - getMissingValue
  - getGeneralSequenceChar
  - getQualitativeCharacters–potentially depending on character features
  - createLeafCharacter
  - missingAligned
- Input.BitPack
  - If new character can be pit packed would go in here
- Functions with "== NonAdd" etc will need extra cases for any new character type

7.1 Execution in Parallel

By default the program will execute multi-threaded based on the number processors available. By specifying the options '+RTS -NX -RTS' where 'X' is the number of processors offered to the program. These are specified after the program as in (for 4 parallel threads):

PhyGraph +RTS -N4 -RTS other options...

Parallel code options are set using a parmap-type strategy throughout the code. THis ujsually speciifed by myParListChunkRDS from the PArallelUtilities module in PhyloLibs The basic definitions of this functionality are found in ParallelUtilities.hs

Acknowledgments

The author would like to thank DARPA SIMPLEX N66001-15-C-4039, the Robert J. Kleberg Jr. and Helen C. Kleberg foundation grant "Mechanistic Analyses of Pancreatic Cancer Evolution", and the American Museum of Natural History for financial support.

8 Bibliography

[1] James S. Farris. Hennig86, Program and Documentation, 1988.

[2] D. S. Gladstein and W. C. Wheeler. POY version 2.0. program and documentation available at http://research.amnh.org/scicomp/projects/poy.php. American Museum of Natural History, New York, 1997.

[3] Pablo A. Goloboff. Techniques for analysing large data sets. In R. DeSalle, G. Giribet, and W. Wheeler, editors, *Techniques in Molecular Systematics and Evolution*, pages 70–79. Birkhäuser Verlag, Basel, 2002.

[4] A. Varón, L. S. Vinh, I. Bomash, and W. C. Wheeler. Poy 4.0. American Museum of Natural History. http://research.amnh.org/scicomp/projects/poy.php, 2008.

[5] A. Varón, L. S. Vinh, and W. C. Wheeler. POY version 4: Phylogenetic analysis using dynamic homologies. *Cladistics*, 26:72–85, 2010.

[6] W. C. Wheeler. *Systematics: A course of lectures*. Wiley-Blackwell, 2012.

[7] W. C. Wheeler, D. S. Gladstein, and J. De Laet. POY version 3.0. program and documentation available at http://research.amnh.org/scicomp/projects/poy.php (current version 3.0.11). documentation by D. Janies and W. C. Wheeler. commandline documentation by J. De Laet and W. C. Wheeler. American Museum of Natural History, New York, 1996-2005.

[8] W. C. Wheeler, N. Lucaroni, L. Hong, L. M. Crowley, and A. Varón. POY version 5.0. American Museum of Natural History. http://research.amnh.org/scicomp/projects/poy.php, 2013.

[9] W. C. Wheeler, N. Lucaroni, L. Hong, L. M. Crowley, and A. Varón. POY version 5: Phylogenetic analysis using dynamic homologies under multiple optimality criteria. *Cladistics*, 31, 2015. 189-196.

[10] W. Timothy J. White and Barbara R. Holland. Faster exact maximum parsimony search with XMP. *Bioinformatics*, 27(10):1359–1367, 03 2011.