Sequence Bloom Tree v0.3 User Guide

April 27, 2015

1 Synopsis

bt hashes [-k 20] hashfile nb_hashes
bt count hashfile bf_size fasta_in filter_out.bf.bv
bt build [–sim-type 0] hashfile filterlistfile outfile
bt compress bloomtreefile outfile
bt check bloomtreefile
bt draw bloomtreefile out.dot
bt query [–max-filters 1] [-t 0.8] [–leaf-only 0] bloomtreefile queryfile outfile
bt sim [–sim-type 0] hashfile bvfile1 bvfile2

2 Analysis Pipeline

To build and query a dataset using SBT, you should follow this general pipeline:

1. Initialize hash functions and user settings by running "hashes" command
2. Convert each fasta file in the database to a bloom filter bit vector using the "count" command.
3. Compile a list of all the bit vectors generated and build the SBT for that set using the "build" command.
4. Compress the bit vectors that make up the entire tree using the built in "compress" command.
5. Save your input queries as a line-separated text file and run the "query" command with the desired threshold and using the compressed SBT file.

3 Description

3.1 Hashes

bt hashes [-k 20] hashfile nb_hashes

- **hashfile** is the location of the file being written
- **nb_hashes** is an integer that sets the number of hashes generated for the bloom filters

Usage:
To build a set of conserved hash functions for the bloom filters, use a command like:

bt hashes myhashfile.hh 1

This will write a file 'myhashfile.hh' which stores the necessary information for the Jellyfish library's bloom filter functions.

3.2 Count

bt count hashfile bf_size fasta_in filter_out.bf.bv

- **hashfile** is the location of the hashfile written using the "hashes" function
- **fasta_in** is the location of the input fasta being counted
- **filter_out.bf.bv** is the location of the bloom filter being written

Usage:
To convert a fasta short-read file to a SBT bit vector, use a command like:

bt count myhashfile.hh 2000000000 SRR001.fasta SRR0001.bf.bv

This will count and hash the k-mers associated with 'SRR001.fasta' using the settings defined by 'myhashfile.hh' and store all k-mers in 'SRR0001.bf.bv'

3.3 Build

bt build [–sim-type 0] hashfile filterlistfile outfile

- **sim-type** is an option that defines the similarity metric used. (0) uses the default Hamming distance between two bit vectors while (1) uses a Jaccard index metric.
- **hashfile** is the location of the hashfile written using the "hashes" function
- **filterlistfile** is the location of a plaintext file containing the paths to all the bit vectors generated by the "count" function
- **outfile** is the location of the SBT structure file being written

Usage:
To build the bloomtree from a list of SBT bit vectors, use a command like:

bt build myhashfile.hh mybitvectorlist.txt mySBT.bloomtree

This will build the SBT through single-threaded insertions of each element in 'mybitvectorlist.txt' and write the union filters to the same directory as the leaves. Once the tree is completely built, the edge-relationships that define the tree will be saved to 'mySBT.bloomtree'.

3.4 Compress

bt compress bloomtreefile outfile

- **bloomtreefile** is the location of the SBT structure file written by the "build" function
- **outfile** is the location of the [compressed] SBT structure file being written

Usage:

To compress the bloomtree from bit vectors to rrr compressed vectors, use a command like:

bt compress mySBT.bloomtree myCompressedSBT.bloomtree

This will compress every file in the original SBT and write a new bloomtree using the same edge-relationships but the rrr compressed files.

3.5 Check

bt check bloomtreefile

- **bloomtreefile** is the location of the SBT structure file written by the "build" function

Usage:

To check that the tree was built and saved correctly such that every parent is the union of its two children, run the built-in check function:

bt check mySBT.bloomtree

This will write a verbose set of output for each parent-child relationship in the function.

3.6 Draw

bt draw bloomtreefile out.dot

- **bloomtreefile** is the location of the SBT structure file written by the "build" function
- **out.dot** is the location of the graphvis file being written

Usage:

To graph the existing bloom tree structure, use a command like:

bt draw mySBT.bloomtree mySBTstructure.dot

This will construct a .dot relationship of all nodes and edges that can be plotted using any number of graph visualization packages.

3.7 Query

bt query [–max-filters 1] [-t 0.8] [–leaf-only 0] bloomtreefile queryfile outfile

- **max-filters** is an option that defines the total number of filters that can be loaded at one time into memory. As filters are loaded only once per query, one filter is usually sufficient for single-threaded operations.

- **threshold (t)** is a float between 0 and 1 that defines the proportion of query k-mers that must be present in any bloom filter to define a "hit". The default value assumes a valid hit contains 80% of exact-matching k-mers.
- **leaf-only** has two possible values. (0) is the default value and searches the entire SBT while (1) ignores the tree structure and queries just the leaf nodes of the tree in a naive search.
- **bloomtreefile** is the location of the SBT structure file written by the "build" function
- **queryfile** is the location of a text file containing line-separated full-length sequences.
- **outfile** is the location of the [compressed] SBT structure file being written

Usage:

To query the SBT for an arbitrary set of sequences, use a command like:

bt query -t 0.8 mySBT.bloomtree myQueryFile.txt myOutFile.txt

This will batch query the bloom tree encoded by 'mySBT.bloomtree' for every line-separated sequence in 'myQueryFile.txt' at a query k-mer threshold of 0.8. If your query of interest is a housekeeping gene or is known to be expressed in the majority of files, it may be beneficial to set the 'leaf_only' option to 1 and ignore the tree structure by querying only the tree leaves.

3.8 sim

bt sim [–sim-type 0] hashfile bvfile1 bvfile2

- **sim-type** is an option that defines the similarity metric used. (0) uses the default Hamming distance between two bit vectors while (1) uses a Jaccard index metric.
- **hashfile** is the location of the hashfile written using the "hashes" function
- **bvfile1, bvfile2** are any combination of two SBT bit vectors constructed using the "count" function.

Usage:

To test the raw bit similarity between two bloom filters encoded by the SBT, use a command like:

bt sim hashfile.hh SRR0001.bf.bv SRR0002.bf.bv

This will return the similarity using either the default Hamming (0) or Jaccard (1) metrics.