TIGER (Timing Inferred from Genome Replication) infers DNA replication timing profiles from whole-genome sequence data of proliferating cell samples.

DNA replication leads to dynamic changes in DNA copy number during the S phase of the cell cycle. Since different parts of the genome replicate at different times, S phase cells have non-uniform DNA copy numbers along their chromosomes. Sequencing genomic DNA from a biological specimen that contains a sufficient fraction of cells in S phase enables the observation of such copy number fluctuations and the inference of the replication timing profile of that specimen. TIGER is a computational pipeline that analyze DNA copy number in whole-genome sequence data to infer these replication timing profiles.

TIGER involves several steps: the counting of sequence reads in genomic windows of a specified number of uniquely alignable basepairs; correction for GC-content effects on sequencing read coverage; removal of copy number variants (CNVs) and alterations (CNAs), repetitive sequences and other technical influences on copy number estimation; and spatial smoothing and normalization to yield final DNA replication timing profiles.

TIGER is composed of two main scripts, alongside several auxiliary functions. It is currently designed for short-read next-generation sequences, is written in Matlab (one process also written in python), and uses BWA-MEM for sequence alignment (other alignment algorithms can also be used) and Samtools for extraction of read locations and alignment scores.

The main scripts in TIGER are: “TIGER\_generate\_processing\_files”, which is run once per reference genome and generates files required for sequence alignability, GC bias calculations, and genomic windows in which sequence reads are counted; and “TIGER\_generate\_replication\_profiles”, which is run per sample (or set of samples) to count sequence reads, correct for GC biases, remove confounders and generate smoothed replication profiles. TIGER differs from other pipelines for DNA copy number inference in that it uses procedures and parameters optimized for inference of DNA replication timing in proliferating cell samples on a background of other biological and technical influences on sequence read coverage.

TIGER\_generate\_processing\_files

**1. Generating an alignability filter**

**a.** Generate fastq file with all possible reads in the genome of a chosen read length:

Load a reference genome- same as used to align sequence files.

Choose read length- equal or longer than the length of reads in the actual sequence files, although there’s no benefit for chosing read lengths greater than 100.

This code generates a zipped fastq file with all possible sequences (“reads”) of the chosen length (excluding gaps). Besides the sequences and coordinates, other metadata saved are arbitrary and identical for all reads.

**b.** Align fastq file using bwa-mem (or alternative aligners)

**c.** Samtools to extract read locations, filter for non-unique alignments (mapQ==0). Uses this command:

samtools view -F 4 -F 16 -F 1024 -q 1 <bam file> | cut -f 4 <file\_name.txt>

-F 4: read unmapped

-F 16: read reverse strand (only the forward strand was written to fastq)

-F 1024: PCR or optical duplicate

-q 1: only keep sequences with mapQ of 1 or more

-f 4: only extract the start position of the sequence

**d.** Import samtools output to matlab

Generates Coordinates\_to\_removeXX file with all genome coordinates that did not pass the alignability filter. Coordinates refer to the first bp of a read.

**2. Generate read number windows**

Based on the alignability filter above (hence this file too is specific to a given read length), and a chosen window size, generate windows with an equal number of uniquely alignable bps.

Windows that span gaps in the reference genome are removed.

Samples sequenced to greater depths could be analyzed using shorter windows, thus increasing resolution without compromising data accuracy. The default window size is 10Kb, although windows as short as 1Kb could be used with deeper (e.g. 30x for a human-sized genome) sequenced libraries and in general provide greater power for identification and removal of short CNVs/CNAs. Shorter windows increase file size and computational time, although typically to manageable levels. Shorter windows can later be merged to larger ones and/or to sliding windows with chosen overlap extent (the function “TIGER\_merge\_windows” supports these operations), however large windows cannot be retrospectively partitioned into smaller ones.

**3. Generate GC content correction files**

Generates files needed for (a) calculating a “GC-bias factor” (the number of reads per GC bin) and (b) calculating the expected number of reads per read number window (“Reads\_expected\_nominal”).

Calculates GC content in bins of 401bp (200bp on each side of each bp in the genome). This bin size is designed to correct for GC biases affecting sequence reads, which are generally 300-500bp long.

**a.** Record genomic coordinates belonging to each GC content bin.

Load reference genome and calculate GC% for 401bps surrounding each bp in the genome, excluding alignability-filtered bps and windows that contain Ns.

For each of the 401 GC content bins, record all genomic coordinates within that bin, save as “hg38\_100bp\_GC\_cont\_401\_filtered\_bins\_chr1”, changing “hg38” (genome build), “100bp” (read length) and “chr1” (chromosome) as appropriate.

**b.** For each read number window, count the number of bps belonging to each of the 401 GC bins (“Reads\_expected\_nominal”).

An optional section enables generating Reads\_expected\_nominal for additional read number window lengths, without running part [a] (which is itself independent of the read number window length).

**TIGER\_generate\_replication\_profiles**

This code: (1) takes read coordinates extracted from a Bam file using Samtools (with appropriate filters), (2) applies the alignability filter, (3) finds reads with outlying copy number for excluding from the GC bias factor calculation (“GC filter”), (4) counts the number of GC-filtered reads in each GC content bin, as well as the total number of bps in each bin after applying the GC filter, (5) calculates the GC bias factor as the ratio of the former two numbers, (6) applies this bias factor per GC bin on each read number window, and corrects for the total coverage of the sequencing library (based on the median number of reads per basepair) to bring the average copy number to 2 (for a diploid genome), (7) divides the read counts per read number window by the expected number of counts, (8) filters for copy number outliers, smoothes the copy number profiles, and normalizes the data values to standard deviation units.

**1. Align and samtools**

Align using bwa-mem (or alternative aligners)

Run: samtools view -F 1024 -F 256 -F 128 -q 10 <bam file> | cut -f 4 <file\_name.txt>

-F 256: not primary alignment

-F 1024: PCR duplicate

-F 128: second in pair (i.e. only extract the locations of the first read in a given pair)

-q 10: only keep sequences with mapQ of 10 or more

-f 4: only extract the start position of the sequence

**2. Apply alignability filter**

Import sam\_ folder with .txt files to matlab, apply alignability filter, save. The code also counts reads in the defined windows along chromosomes, and performs segmentation on the data, which will be used for the GC correction.

An optional section enables changing the segmentation parameters, e.g. for low coverage libraries.

**3. Define GC filter**

For calculating the GC bias factor, use the data segmentation to only take reads within regions of copy number close to 2 (for a diploid genome), i.e. don’t take reads within outlier copy number regions, as these may skew the calculation. Make a variable with all bp coordinates within these regions (this is the GC filter).

**4. Count reads in each GC bin**

Load files with all genomic coordinates belonging to each GC bin (e.g. “hg38\_100bp\_GC\_cont\_401\_filtered\_bins\_chr1”), apply GC filter, then count how many reads in the sequence data belong to each GC bin, and how many total bps in the genome belong to each bin (for normalization).

**5. Generate GC bias factor**

The GC bias factor is the fraction of reads in each GC bin, divided by fraction of bps in each bin.

GC bias correction is applied only on bins of 20-80% GC (bins 81 to 321 of the 401 bins).

**6. Calculate expected number of reads per window**

Based on “Reads\_expected\_nominal” (step 3 in TIGER\_generate\_processing\_files), multiply the number of bps of each GC bin in each read number window by the corresponding GC bias factor (note that the GC filter is not applied here- the GC correction is applied on all reads). Then, multiply by the median number of reads per basepair, divided by2(to make the genome diploid after normalization)- this is the expected number of reads per window.

**7. Normalize read counts by expected**

Divide read counts per read number window (from step 3) by the expected count (from step 6). This gives the “normalized” count, which is raw DNA copy number (or replication timing) data.

The code also saves the individual count and expected values. These are useful, e.g. for calculating total number of reads (or coverage) of the library, or for merging windows into larger windows.

**8. Filter, smooth, normalize**

Filter for copy number variants, alterations and outliers using TIGER\_segment\_filt (parameters may need to be adjusted per sample).

Smooth the raw data (using the Matlab csaps function with default paremeter=10-17) over genomic regions between gaps in the reference sequence or the data.

Normalize the data to mean=0 and std=1, i.e. z-score or standard deviation units.

**TIGER\_segment\_filt**

Function to filter out copy number variants, alterations, and other copy number outliers based on segmentation. Used twice in the code: for defining the GC filter, and for filtering the raw copy number data before smoothing.

Uses the Matlab function segment with correction for the copy number value of segments such that the segments accurately correspond to the underlying data. This correction is achieved by re-calculating the mean value of each segment based directly on the data. To avoid mis-calling segment ends (such that a segment extends into data points that should belong to the next segment), a relatively low segmentation R2 value should be chosen (typically <1).

**TIGER\_merge\_windows**

Function that takes Count\_expected\_data and window locations, number of windows to merge and genome build; merges window locations into larger windows and provides a new Count\_expected\_data and corresponding locations. It then removes merged windows that span gaps in the reference genome sequence.