DNAscent

Release 2.0.2

Sep 06, 2021

Contents:

CHAPTER 1

Download & Installation

Development was done using gcc 9.3.0 on an Ubuntu 16.04 platform. While installation on other platforms is possible, Ubuntu is the platform that is recommended and supported.

Clone the DNAscent repository with the recursive flag so that the dependencies are cloned as well.

```
git clone --recursive https://github.com/MBoemo/DNAscent.git
```

The DNAscent directory will appear in your current directory. Switch to the latest release and compile the software by running:

```
cd DNAscent
git checkout 2.0.2
make
```

This will put the DNAscent executable into the DNAscent/bin directory. Compilation requires a version of gcc that supports C++14, and a typical compile time for DNAscent and all of its dependencies is 5-7 minutes.

Cloning the repository recursively (see above) will provide all the required dependencies so you don't need to find them yourself. For completeness, however, they are listed here:

- pfasta (https://github.com/kloetzl/pfasta)
- fast5 (https://github.com/mateidavid/fast5.git)
- htslib (https://github.com/samtools/htslib.git)
- hdf5lib (https://support.hdfgroup.org/HDF5/)
- tinydir (https://github.com/cxong/tinydir.git)
- TensorFlow (https://www.tensorflow.org/install/lang_c)

Please note that the high throughput sequencing library (htslib) requires bzlib and lzma for compression. While these are common on most systems, if you don't have these, apt-get lzma-dev, liblzma-dev, and libbz2-dev. In addition, pfasta requires libbsd on Linux.

1.1 VBZ Fast5 Compression

In new versions of MinKNOW, the fast5 files are compressed with VBZ Compression (see https://github.com/nanoporetech/vbz_compression). To use DNAscent on these compressed fast5 files, do the following (N.B., we're assuming you don't have root permissions):

1. Go to https://github.com/nanoporetech/vbz_compression/releases and download the plugin appropriate for your processor architecture. In this example, we'll use `ont-vbz-hdf-plugin-1.0.1-Linux-x86_64.tar.gz`.
2. Download and unpack the plugin:

```
wget https://github.com/nanoporetech/vbz_compression/releases/download/v1.0.1/ont-
↪vbz-hdf-plugin-1.0.1-Linux-x86_64.tar.gz
tar -xf ont-vbz-hdf-plugin-1.0.1-Linux-x86_64.tar.gz
```

3. Add the plugin to your path:

```
export HDF5_PLUGIN_PATH=/full/path/to/ont-vbz-hdf-plugin-1.0.1-Linux/usr/local/
↪hdf5/lib/plugin
```

4. Run `DNAscent detect` as normal.

1.2 GPU Use

The `DNAscent detect` executable can make use of a GPU, although this is optional (see *Detect*). DNAscent requires CUDA 10.0 and cuDNN 7.5, and information about these can be found at the following links:

- cuDNN: https://developer.nvidia.com/cudnn
- CUDA: https://developer.nvidia.com/cuda-10.0-download-archive

Always discuss any installation or version changes with your system administrator.

CHAPTER 2

Index

`DNAscent index` is a `DNAscent` subprogram that creates a map between Oxford Nanopore readIDs and fast5 files. This allows `DNAscent detect` to scan through bam files and pull out the relevant signal information for each read.

2.1 Usage

```
To run DNAscent index, do:
   DNAscent index -f /path/to/fast5Directory
Required arguments are:
  -f,--files                path to fast5 files.
Optional arguments are:
  -o,--output               output file name (default is index.dnascent),
  -s,--sequencing-summary   path to sequencing summary file Guppy (optional but␣
↪strongly recommended).
```

The only required input to `DNAscent index` is the full path to the top-level directory containing the sequencing run's fast5 files, passed using the `-f` flag. This will typically be the directory created with MinKNOW during sequencing. An additional optional argument is the full path to the `sequencing_summary.txt` file, specified using the `-s` flag. This file is created by Guppy during basecalling and is located in the top level directory containing the Guppy-created fastq files. While including the sequencing summary file is optional, it is strongly recommended as it will make `DNAscent index` run much faster. The default behaviour of `DNAscent index` is to place a file called `index.dnascent` in the working directory. The name of this file can be overridden using the `-o` flag.

2.2 Output

`DNAscent index` will put a file called `index.dnascent` in the current working directory (note that if you used the `-o` flag, then the file will have the name and location that you specified). This file will be needed as an input to `DNAscent detect`.

CHAPTER 3

Detect

`DNAscent detect` is a `DNAscent` subprogram that goes through each read and, at each thymidine position, assigns the probability the thymidine is BrdU.

3.1 Usage

```
To run DNAscent detect, do:
   DNAscent detect -b /path/to/alignment.bam -r /path/to/reference.fasta -i /path/to/
↪index.dnascent -o /path/to/output.detect
Required arguments are:
  -b,--bam                  path to alignment BAM file,
  -r,--reference            path to genome reference in fasta format,
  -i,--index                path to DNAscent index,
  -o,--output               path to output file that will be generated.
Optional arguments are:
  -t,--threads              number of threads (default is 1 thread),
  --GPU                      use the GPU device indicated for prediction (default is␣
↪CPU),
  -q,--quality              minimum mapping quality (default is 20),
  -l,--length               minimum read length in bp (default is 100).
```

The main input of `DNAscent detect` is an alignment (bam file) between the sequence fastq from Guppy and the organism's reference genome. This bam file should be sorted using `samtools sort` and indexed using `samtools index` so that there is a .bam.bai file in the same directory as the bam file. (Please see the example in *Workflow* for details on how to do this.) The full path to the reference genome used in the alignment should be passed using the `-r` flag, and the index required by the `-i` flag is the file created using `DNAscent index` (see *Index*).

The number of threads is specified using the `-t` flag. `DNAscent detect` multithreads quite well by analysing a separate read on each thread, so multithreading is recommended. By default, the signal alignments and ResNet BrdU predictions are run on CPUs. If a CUDA-compatible GPU device is specified using the `--GPU` flag, then the signal alignments will be run on CPUs using the threads specified with `-t` and the ResNet BrdU prediction will be run on the GPU. Your GPU device number can be found with the command `nvidia-smi`. GPU use requires that CUDA

and cuDNN are set up correctly on your system and that these libraries can be accessed. If they're not, DNAscent will default back to using CPUs.

It is sometimes useful to only run `DNAscent detect` on reads that exceed a certain mapping quality or length threshold (as measured by the subsequence of the contig that the read maps to). In order to do this without having to filter the bam file, DNAscent provides the `-l` and `-q` flags. Any read in the bam file with a reference length lower than the value specificed with `-l` or a mapping quality lower than the value specified with `-q` will be ignored.

Before calling BrdU in a read, `DNAscent detect` must first perform a fast event alignment (see https://www.biorxiv.org/content/10.1101/130633v2 for more details). Quality control checks are performed on these alignments, and if they're not passed, then the read fails and is ignored. Hence, the number of reads in the output file will be slightly lower than the number of input reads. Typical failure rates are about 5-10%, although this will vary slightly depending on the read length, the BrdU substitution rate, and the genome sequenced.

3.2 Output

`DNAscent detect` will produce a single human-readable output file with the name and location that you specified using the `-o` flag. To aid organisation and reproducibility, each detect file starts with a short header. The start of each header line is always a hash (#) character, and it specifies the input files and settings used, as well as the version and commit of DNAscent that produced the file. An example is as follows:

```
#Alignment /path/to/alignment.bam
#Genome /path/to/reference.fasta
#Index /path/to/index.dnascent
#Threads 1
#Compute CPU
#Mode CNN
#MappingQuality 20
#MappingLength 5000
#SignalDilation 1.000000
#Version 2.0.0
#Commit 4cf80a7b89bdf510a91b54572f8f94d3daf9b167
```

You can easily access the header of any .detect file with `head -11 /path/to/output.detect` or, alternatively, `grep '#' /path/to/output.detect`.

Below the header is data for each read. Note that everything in this output file orients to the reference genome in the 5' –> 3' direction. Each read starts with a line in the format:

```
>readID contig mappingStart mappingEnd strand
```

These lines always begin with a greater-than (>) character. Therefore, an easy way to count the number of reads in the file is `grep '>' detect.out | wc -l`. The fields are:

- `readID` is a unique hexadecimal string assigned to each read by the Oxford Nanopore software,
- the read mapped between `mappingStart` and `mappingEnd` on `contig`,
- `strand` either takes the value `fwd`, indicating that the read mapped to the forward strand, or `rev` indicating that the read mapped to the reverse complement strand.

The following shows an example for a read that to the reverse strand between 239248 and 286543 on chrII.

```
>c602f23f-e892-42ba-8140-da949abafbdd chrII 239248 286543 rev
```

Below these "start of read" lines, each line corresponds to the position of a thymidine in that read. There are three tab-separated columns:

- the coordinate on the reference,
- probability that the thymidine is actually BrdU,
- 6mer on the reference.

Consider the following examples:

```
>c6785e1f-10d2-49cb-8ca3-e8d48979001b chrXIII 74003 81176 rev
74010   0.012874        TCTCTA
74011   0.012428        CTCTAA
74014   0.016811        TAACGA
74017   0.013372        CGACCA
74018   0.013836        GACCAA
```

Here, we're looking at the sequence TCTCTAACGACCAA on the reference genome. Because this read maps to the reverse complement, a call is made at every A (instead of T) on the reference. If instead we looked at a read that mapped to the forward strand, an example would be:

```
>5d10eb9a-aae1-4db8-8ec6-7ebb34d32575 chrXIII 72319 77137 fwd
72319   0.017496        TCGTTT
72322   0.029483        TTTCTG
72323   0.039008        TTCTGT
72324   0.031474        TCTGTG
72326   0.026997        TGTGAG
```

In both of these output snippets, we see from the second column that the probability of BrdU is low (around a 1-3% chance of BrdU) so these few bases are likely from a BrdU-negative region of DNA. In contrast, here we see the start of a read that does contain BrdU, and accordingly, the probability of BrdU at some positions is much higher:

```
>a4f36092-b4d5-47a9-813e-c22c3b477a0c chrXVI 899273 907581 fwd
899276  0.866907        TCAAAT
899281  0.947935        TCCACA
899300  0.014683        TGGGAG
899312  0.186812        TAACGG
899320  0.934850        TTATTG
```

CHAPTER 4

Regions

`DNAscent regions` is a `DNAscent` subprogram that interprets the output of `DNAscent detect` to call regions of high and low BrdU incorporation.

Note that as of v2.0, `DNAscent regions` has been largely superceded by `DNAscent forkSense` and the increased accuracy of `DNAscent detect` makes visualising BrdU incorporation in regions mostly unnecessary. However, it is still included to avoid breaking legacy workflows, and it does still have some uses as explained below.

4.1 Usage

```
To run DNAscent regions, do:
   DNAscent regions -d /path/to/DNAscentOutput.detect -o /path/to/DNAscentOutput.
↪regions
Required arguments are:
  -d,--detect               path to output file from DNAscent detect,
  -o,--output               path to output directory for bedgraph files.
Optional arguments (if used with default ResNet-based detect) are:
  -r,--resolution           number of thymidines in a region (default is 10).
Optional arguments (if used with HMM-based detect) are:
     --threshold            probability above which a BrdU call is considered␣
↪positive (default: 0.8),
  -c,--cooldown             minimum gap between positive analogue calls (default: 4),
  -r,--resolution           minimum length of regions (default is 100 bp),
  -p,--probability          override probability that a thymidine 6mer contains a␣
↪BrdU (default: automatically calculated),
  -z,--zScore               override zScore threshold for BrdU call (default:␣
↪automatically calculated).
```

The only required input of `DNAscent regions` is the output file produced by `DNAscent detect`. `DNAscent regions` will first look through the detect file and determine the approximate fraction of thymidines replaced by BrdU in BrdU-positive regions. Using this probability, a z-score is assigned to each window (100 bp wide by default, but this can be changed using the `-r` flag) to indicate whether there is more or less BrdU than would be expected for an average BrdU-positive region. Naturally, some regions will be BrdU-positive but will have a substitution rate

lower than average for BrdU-positive regions. Hence, `DNAscent regions` determines an appropriate boundary threshold between BrdU-positive regions and thymidine regions and rescales all of the z-scores so that this boundary is 0. `DNAscent regions` will calculate these values for you, but they can be overridden with the `-p` and `-z` flags, though this is generally not recommended. The exceptions are runs with 0% BrdU or runs where a high BrdU incorporation is expected along the entirety of each read. This is because these parameters are computed assuming that there are two populations (BrdU-positive and thymidine-only segments of DNA).

In order to determine regions of high and low BrdU incorporation, `DNAscent regions` needs to count positive BrdU calls. By default, a thymidine is considered to be BrdU if it was scored with a probability higher than 0.8 by `DNAscent detect`. This value was tuned in-house to optimise signal-to-noise, but it can be changed with the `--threshold` flag. Likewise, some care has to be given to how positive calls are counted, as BrdU can sometimes shift the signal of neighbouring thymidines. To prevent artefacts from overcounting while minimising undercounting, the default behaviour is to only make a positive call at most every 4 bases, though this can be changed with the `-c` flag.

4.2 Output

The output of DNAscent regions is a file with similar formatting to that of `DNAscent detect`. The format for the read headers is the same. From left to right, the tab-delimited columns indicate:

- the start of the region,
- the end of the region,
- the z-score,
- the string "BrdU" if the score is positive and "Thym" if the score is negative.

A large positive z-score indicates high BrdU incorporation in that region, and a large negative score indicates very little BrdU incorporation in that region. An example output is as follows:

```
>bfdc06e0-001f-41f7-bbea-f2f6785a3860 chrI 0 28066 fwd
62      167     -2.38086        Thym
173     276     -2.38086        Thym
283     388     -2.27466        Thym
393     499     -2.00741        Thym
501     605     -2.00741        Thym
606     708     -2.48397        Thym
713     817     -2.27466        Thym
```

Note that the region width may sometimes vary slightly from the value specified. The region width is designated as the coordinate of the first thymidine greater than the window width (100 bp by default) from the starting coordinate. In order to guard against assigning a score to regions with very few thymidines, `DNAscent regions` will also extend the region until at least 10 calls are considered.

CHAPTER 5

forkSense

`DNAscent forkSense` is a `DNAscent` subprogram that provides a probability estimate at each thymidine that a leftward- or rightward-moving fork moved through that position during the BrdU pulse.

5.1 Usage

```
To run DNAscent forkSense, do:
   DNAscent forkSense -d /path/to/BrdUCalls.detect -o /path/to/output.forkSense
Required arguments are:
  -d,--detect               path to output file from DNAscent detect,
  -o,--output               path to output file for forkSense.
Optional arguments are:
  -t,--threads              number of threads (default: 1 thread),
     --markOrigins             writes replication origin locations to a bed file␣
↪(default: off),
     --markTerminations        writes replication termination locations to a bed file␣
↪(default: off),
     --markForks               writes replication fork locations to a bed file␣
↪(default: off).
```

The only required input of `DNAscent forkSense` is the output file produced by `DNAscent detect`. Note that the detect file must have been produced using the v2.0 ResNet algorithm; `DNAscent forkSense` is not compatible with legacy HMM-based detection.

If the `--markOrigins` flag is passed, `DNAscent forkSense` will use detected leftward- and rightward-moving forks to infer the locations of fired replication origins and write these to a bed file called `origins_DNAscent_forkSense.bed` in the working directory. Likewise, if the `--markTerminations` flag is passed, termination sites will be recorded in a bed file called `terminations_DNAscent_forkSense.bed`.

5.2 Output

If `--markOrigins` and/or `--markTerminations` were used, the resulting bed files has one called origin (for origins_DNAscent_forkSense.bed) or termination site (for terminations_DNAscent_forkSense.bed) per line and, in accordance with bed format, have the following space-separated columns:

- chromosome name,
- 5' boundary of the origin (or terminiation site),
- 3' boundary of the origin (or terminiation site),
- read header of the read that the call came from (similar to those in the output file of `DNAscent detect`).

Note that the "resolution" of the calls (i.e., the third column minus the second column) will depend on your experimental setup. In synchronised early S-phase cells, this difference for origin calls is likely to be small as the leftward- and rightward-moving forks from a fired origin are nearby one another. In asynchronous or mid/late S-phase cells, the difference is likely to be larger as the forks from a single origin will have travelled some distance before the BrdU pulse. The bed files only specify the region between matching leftward- and rightward-moving forks. Any subsequent assumptions (such as assuming uniform fork speed and placing the origin in the middle of that region) are left to the user.

The output of `DNAscent forkSense` is a file with similar formatting to that of `DNAscent detect`. The format for the read headers is the same. From left to right, the tab-delimited columns indicate:

- the coordinate on the reference,
- probability that a leftward-moving fork passed through that coordinate during a BrdU pulse,
- probability that a rightward-moving fork passed through that coordinate during a BrdU pulse.

A low probability in both the second and third columns suggests it was unlikely that a fork passed through that position during the pulse.

The following example output shows the end of a read that was passed through by a leftward-moving fork:

```
>22c8a674-ed0e-475f-9c54-cb185299d923 chrII 173332 210452 fwd
173339  0.687217        0.062620
173341  0.687217        0.062620
173342  0.687217        0.062620
173345  0.687217        0.062620
173347  0.687217        0.062620
173348  0.687217        0.062620
173349  0.743986        0.045767
173358  0.743986        0.045767
173375  0.743986        0.045767
173377  0.743986        0.045767
173378  0.743986        0.045767
173381  0.743986        0.045767
173382  0.806924        0.038138
173383  0.806924        0.038138
173387  0.806924        0.038138
173390  0.806924        0.038138
173392  0.806924        0.038138
173393  0.806924        0.038138
173398  0.846875        0.032027
173402  0.846875        0.032027
173404  0.846875        0.032027
173406  0.846875        0.032027
173407  0.846875        0.032027
```

(continues on next page)

(continued from previous page)

```
173417  0.846875        0.032027
173418  0.906748        0.028587
173419  0.906748        0.028587
173423  0.906748        0.028587
173425  0.906748        0.028587
173426  0.906748        0.028587
173428  0.906748        0.028587
173441  0.909755        0.029341
173445  0.909755        0.029341
173446  0.909755        0.029341
173449  0.909755        0.029341
173450  0.909755        0.029341
173451  0.909755        0.029341
173454  0.907803        0.029983
```

CHAPTER 6

psl

`DNAscent psl` is a `DNAscent` subprogram that writes a psl file to visualise the output of `DNAscent detect`.

6.1 Usage

```
To run DNAscent psl, do:
   DNAscent psl -d /path/to/DNAscentOutput.detect -r /path/to/reference.fasta -o /
↪path/to/psl_prefix
Required arguments are:
  -d,--detect               path to output file from DNAscent detect,
  -r,--reference            path to genome reference in fasta format,
  -o,--output               path to output bed prefix.
Optional arguments are:
     --threshold            probability above which a BrdU call is considered␣
↪positive (default: 0.8),
     --min                  minimum read length to compute (default is 1),
     --max                  maximum read length to compute (default is Inf).
```

The output file from `DNAscent detect` should be passed using the `-d` flag, and the reference genome used in the alignment should be passed with the `-r` flag.

6.2 Output

The output is a psl file with each positive BrdU call marked as a tick. These files can then be opened in IGV or the UCSC Genome Browser to visualise positive BrdU calls genome-wide. Note that psl tracks are only plotted from the location of the first tick, so in order to visualise the portions of each read before the first BrdU call and after the last BrdU call, a placeholder tick is placed at the first and last coordinate of each read.

CHAPTER 7

Visualisation

DNAscent supports multilevel analysis: We want users to be able to see the fork calls made by `DNAscent forkSense` and visualise them alongside individual base-pair resolution BrdU calls by `DNAscent detect` in order to see why these calls are being made. To that end, we include a visualisation utility in `DNAscent/utils` that formats the output of DNAscent executables (detect, regions, and forkSense) into bedgraphs that can be visualised with IGV or the UCSC Genome Browser. You can supply this utility with the output from one, two, or all three of these executables. If more than one is specified, the utility organises the bedgraphs so that the tracks for each read are grouped together.

7.1 Usage

```
dnascent2bedgraph.py: Converts the output of DNAscent detect, regions, and forkSense␣
↪into bedgraphs.
To run dnascent2bedgraph.py, do:
  python dnascent2bedgraph.py [arguments]
Example:
  python dnascent2bedgraph.py -d /path/to/dnascentDetect.out -f /path/to/
↪dnascentForksense.out -o /path/to/newBedgraphDir -n 1000 --minLength 10000
Required arguments are at least one of the following:
  -d,--detect               path to DNAscent detect output file,
  -f,--forkSense            path to DNAscent forkSense output file,
  -r,--regions              path to DNAscent regions output file.
Required argument is:
  -o,--output               output directory which will be created.
Optional arguments are:
     --minLength            only convert reads with specified minimum read length (in␣
↪base pairs) into bedgraphs (default: 1),
     --maxLength            only convert reads with specified maximum read length (in␣
↪base pairs) into bedgraphs (default: Inf),
  -n,--maxReads             maximum number of reads to convert into bedgraphs␣
↪(default: Inf),
     --filesPerDir          maximum reads per subdirectory (default: 300).
```

A further example of how to use `dnascent2bedgraph` is given in *Workflow*.

7.2 Output

`dnascent2bedgraph` will create the directory you specified using the `-o` flag which will contain integer-numbered subdirectories. Each of these subdirectories will contain the bedgraphs for the number of reads specified by `--filesPerDir` (default is 300). If the output of more than one DNAscent executable was specified using the `-d`, `-f`, and `-r` flags, then the bedgraphs for each read will be grouped together so that they appear in IGV as consecutive tracks.

CHAPTER 8

Workflow

The following is a full DNAscent workflow, where we'll start off after Guppy has finished running (users that need help with Guppy should refer to the Oxford Nanopore webpages). In particular, we assume the following:

- you have a directory of 1D R9.5 or R9.4.1 450bp/s Oxford Nanopore fast5 reads (which may be in subdirectories) that you want to use for detection,
- these reads have been basecalled to fastq format using Albacore or Guppy (available from Oxford Nanopore),
- you have a reference/genome file (in fasta format) for your reads.

8.1 Example Workflow

Download and compile DNAscent:

```
git clone --recursive https://github.com/MBoemo/DNAscent.git
cd DNAscent
git checkout 2.0.2
make
cd ..
```

Concatenate the fastq files from Guppy:

```
cat /path/to/GuppyOutDirectory/*.fastq > reads.fastq
```

Align the reads with minimap2 and sort with samtools:

```
minimap2 -ax map-ont -o alignment.sam /path/to/reference.fasta reads.fastq
samtools view -Sb -o alignment.bam alignment.sam
samtools sort alignment.bam alignment.sorted
samtools index alignment.sorted.bam
```

Now we're ready to use DNAscent. Let's index the run:

```
DNAscent index -f /full/path/to/fast5 -s /full/path/to/GuppyOutDirectory/sequencing_
↪summary.txt
```

This should put a file called `index.dnascent` in the current directory.

You can run DNAscent detect (on 10 threads, for example) by running:

```
DNAscent detect -b alignment.sorted.bam -r /full/path/to/reference.fasta -i index.
↪dnascent -o output.detect -t 10
```

Alternatively, if the system has a CUDA-compatible GPU in it, we can run `nvidia-smi` to get an output that looks like the following:

```
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 450.51.06    Driver Version: 450.51.06    CUDA Version: 11.0     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  Tesla P100-PCIE...  On   | 00000000:05:00.0 Off |                    0 |
| N/A   41C    P0    52W / 250W |   2571MiB / 16280MiB |     43%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|    0   N/A  N/A    178943      C   ...DNAscent_dev/bin/DNAscent     2569MiB |
+-----------------------------------------------------------------------------+
```

From this, we can see that the GPU's device ID is 0 (just to the left of Tesla) so we can run:

```
DNAscent detect -b alignment.sorted.bam -r /full/path/to/reference.fasta -i index.
↪dnascent -o output.detect -t 10 --GPU 0
```

Note that we're assuming the CUDA libraries for the GPU have been set up properly (see *Download & Installation*). If these libraries can't be accessed, DNAscent will splash a warning saying so and default back to using CPUs.

When `DNAscent detect` is finished, it will should put a file called `output.detect` in the current directory. We can look at the individual positive BrdU calls with `DNAscent psl`. Let's create a psl file that shows any position where BrdU is called at 0.7 probability or higher:

```
DNAscent psl -d output.detect -r /full/path/to/reference.fasta -o output --threshold␣
↪0.7
```

The resulting file `output.psl` can be loaded into IGV or the UCSC Genome Browser. At this point, we can make bedgraphs out of the `DNAscent detect` output (see *Visualisation*) which can also be loaded into IGV or the UCSC Genome Browser.

Lastly, we can run `DNAscent forkSense` on the output of `DNAscent detect` to measure replication fork movement. Let's run it on four threads and specify that we want it to keep track of replication origins, forks, and termination sites:

```
DNAscent forkSense -d output.detect -o output.forkSense -t 4 --markOrigins --
↪markTerminations --markForks
```

This will make three files: origins_DNAscent_forkSense.bed (with our origin calls), terminations_DNAscent_forkSense.bed (with our termination calls), and output.forkSense. We can load the two bed files directly into IGV to see where origins and terminiations were called in the genome.

We can visualise (see *Visualisation*) the first 1500 reads of output.forkSense by turning them into bedgraphs:

```
python dnascent2bedgraph.py -d output.detect -f output.forkSense -o␣
↪newBedgraphDirectory -n 1500
```

This will create a new directory called `newBedgraphDirectory`. By passing both a `forkSense` and `detect` file to dnascent2bedgraph.py, the utility will convert them both into bedgraphs and organise them so that for each read, we can see the bp-resolution BrdU detection output from `DNAscent detect` right next to the left- and rightward-moving fork probabilities from `DNAscent forkSense`. These bedgraphs can then be loaded into IGV or the UCSC Genome Browser.

8.2 Barcoding

The workflow for a barcoded run is very similar to the workflow above with a few minor changes. If you're using a barcoded run that you demultiplexed with Guppy, make a fastq file for each barcode and align each of them to the reference to make as many bam files as you have barcodes. Then run `DNAscent detect` on the bam file for each barcode. You only have to run `DNAscent index` once per run, and the same `index.dnascent` file can be passed to `DNAscent detect` regardless of which barcode you're working with.

CHAPTER 9

Python Cookbook

The output file formats of all DNAscent executables were specifically designed to be easy to parse with short (Python, Perl, etc.) scripts with the aim of making it simple for users to make application-specific plots. Here, we provide a brief "cookbook" of barebones analysis scripts that can be copied and modified by users.

The following barebones script parses the output of `DNAscent detect`. We iterate line-by-line and parse each field in the file.

```
f = open('path/to/output.detect','r')

for line in f:

     #ignore the header lines
     if line[0] == '#':
             continue

     #split the line into a list by whitespace
     splitLine = line.rstrip().split()

     if line[0] == '>':

             readID = splitLine[0][1:]
             chromosome = splitLine[1]
             refStart = int(splitLine[2])
             refEnd = int(splitLine[3])
             strand = splitLine[4]
     else:
             posOnRef = int(splitLine[0])
             probBrdU = float(splitLine[1])
             sixMerOnRef = splitLine[2]

             #add these values to a container or do some processing here

f.close()
```

The following barebones script parses the output of `DNAscent forkSense`. Note the similarity to the above script:

All DNAscent output files were designed to have a very similar format to aid user processing.

```
f = open('path/to/output.forkSense','r')

for line in f:

     #ignore the header lines
     if line[0] == '#':
             continue

     #split the line into a list by whitespace
     splitLine = line.rstrip().split()

     if line[0] == '>':

             readID = splitLine[0][1:]
             chromosome = splitLine[1]
             refStart = int(splitLine[2])
             refEnd = int(splitLine[3])
             strand = splitLine[4]
     else:
             posOnRef = int(splitLine[0])
             probLeftFork = float(splitLine[1])
             probRightFork = float(splitLine[2])

             #add these values to a container or do some processing here

f.close()
```

And again for `DNAscent regions`:

```
f = open('path/to/output.regions','r')

for line in f:

     #ignore the header lines
     if line[0] == '#':
             continue

     #split the line into a list by whitespace
     splitLine = line.rstrip().split()

     if line[0] == '>':

             readID = splitLine[0][1:]
             chromosome = splitLine[1]
             refStart = int(splitLine[2])
             refEnd = int(splitLine[3])
             strand = splitLine[4]
     else:
             regionStart = int(splitLine[0])
             regionEnd = int(splitLine[1])
             regionScore = float(splitLine[2])

             #add these values to a container or do some processing here

f.close()
```

CHAPTER 10

Release Notes

10.1 v2.0.2

- Migration from HMM-based BrdU detection at every thymidine to ResNet-based detection at every thymidine,
- Significant increases to BrdU detection accuracy,
- Support for BrdU detection on GPUs,
- `DNAscent forkSense` to call replication origins and termination sites in both synchronously and asynchronously replicating cells at any point in S-phase,
- `DNAscent align` to align nanopore signals to reference,
- Significant increases to replication origin calling accuracy and sensitivity,
- Visualisation utility for plotting output of multiple DNAscent executables as bedgraphs,
- Released with Boemo, MA. DNAscent v2: Detecting Replication Forks in Nanopore Sequencing Data with Deep Learning. bioRxiv 2020.

10.2 v1.0.0

- HMM-based BrdU detection at every thymidine,
- Improvements to BrdU detection accuracy,
- `DNAscent train` to train Guassian mixture models from nanopolish eventalign.

10.3 v0.1.0

- HMM-based BrdU detection at ~160 thymidine-containing 6mers,
- Assignment of high- and low-BrdU regions based on Z-score,

- Replication origin calling for early S-phase cells,
- Released with Muller and Boemo, et al. Capturing the dynamics of genome replication on individual ultra-long nanopore sequence reads. Nature Methods 2019;16:429-436.

CHAPTER 11

Overview

DNAscent is software designed to detect the modified base BrdU in Oxford Nanopore reads. In an experimental setup where BrdU is incorporated into nascent DNA by replication forks, this software can be used to answer questions that were traditionally answered by DNA fibre analysis.

DNAscent is under active development by the Boemo Group based in the Department of Pathology, University of Cambridge. We aim to push regular updates and improvements, and incorporating new functionality is an active area of our computational research.

CHAPTER 12

Publications

Please cite the following publication if you use DNAscent for your research:

Boemo, MA. DNAscent v2: Detecting Replication Forks in Nanopore Sequencing Data with Deep Learning. bioRxiv 2020. [Preprint DOI]

Muller CA, Boemo MA, Spingardi P, Kessler, BM, Kriaucionis S, Simpson JT, Nieduszynski CA. Capturing the dynamics of genome replication on individual ultra-long nanopore sequence reads. Nature Methods 2019;16:429-436. [Journal DOI]

CHAPTER 13

Bugs, Questions, and Comments

Should any bugs arise or if you have any questions about usage, please raise a GitHub issue. If you have comments or suggestions to improve the software or the documentation, please Email Michael Boemo at mb915@cam.ac.uk.