VIRAL INTEGRATION SIMULATION MANUAL

Developed by vacation student Susie Grigson (email susie.grigson@flinders.edu.au) to simulate viral integrations for testing with the AAV integrations detection pipeline.

insert_virus.py

Simulates integration of a viral genome into host DNA. Host DNA can be a chromosome, region of interest or genome. Separate FASTA files for the host and virus sequences must be parsed and the number of integrations and episomes specified. The location of output files must also be specified.

The main script consists of a viral integration loop and a viral episome loop. Both of these loops continue to add integrations or episome until the number requested is reached. Several different types of integrations and episomes are possible. By default, the type is randomly selected for each integration or episome. This is achieved by randomly selecting a type of integration or episome from a list of possible options. If only integrations of a specific type are desired, the type of viral integrations can be specified at the command line. This is useful for testing the sensitivity of the pipeline to different types of integrations.

The script relies on generating random values. Changing the seed within the script is an easy way to generate different replicates with the same conditions.

Integrations

All types of integrations exist as integration objects. These are handled by the integrations class. If there a multiple chromosomes included in the host FASTA file, a random chromosome is selected for the integration. Each integration object is randomly assigned an integration point, hPos, on the selected chromosome. This value is checked with existing integration sites to ensure it has not been used for a previous integration. If the integration site has been used previously, a new integration site is generated. Five attempts can be made to generate an integration before abandoning and moving on to the next integration.

Each integration object is assigned a viral 'chunk', a region of viral DNA inserted in the host sequence. Viral chunks are generated by the ViralChunk class. These chunks may consist of the whole viral sequence, a random portion of the viral sequence or a random portion of the viral sequence with a length specified by the user. If the whole sequence is selected, the entire viral sequence becomes the viral chunk. If part of the viral sequence is selected, the viral sequence between two randomly generated points becomes the viral chunk. To prevent insertion of very small

regions of viral DNA (say a few base pairs) the minimum length of an integration can be parsed to the script (min_len). Alternatively, if a viral integration of a specified length is selected, a segment of viral sequence of the specified length is extracted from a random point. This specified length (set_len) must be parsed to the script.

Viral chunks may have forward or reverse orientation. This is randomly generated. The reverse compliment is taken of chunks with reverse orientation. Chunks with forward orientation are unchanged.

Viral chunks may also be modified using the viralChunk class. This enables different types of integrations to occur. These modifications require that the viral Chunk is split into segments. The number of segments is drawn from a poisson distribution with $\lambda = 1.5$. The viral chunk is then randomly split into the determined number of segments and the resulting segments stored in a dictionary. If a deletion is required, a randomly selected segment which is not positioned at either end of the chunk is deleted. Alternatively, if a rearrangement is required, segments in the chunk dictionary are assembled in a random order. After modification, the segments are concatenated and integrated in the host sequence.

The methods described in the viralChunk class enable several different types of integrations (Table 1).

Table 1: Types of integrations included in *insert_virus.py*

Name	**Description**
InsertWholeVirus	Inserts the entire viral sequence.
insertViralPortion	Inserts a random portion of viral DNA with random length.
insertSetLength	Inserts a random portion of viral DNA with a predetermined length. Requires set_len be parsed to the script. This is not included if random integrations are generated. This could be useful for applications including detecting viral integrations with k-mer barcoding.
InsertWholeRearrange	Inserts the viral sequence with a rearrangement.
InsertWholeDeletion	Inserts the viral sequence with a deletion.

insertPortionRearrange	Inserts a portion of the viral sequence with a rearrangement.
insertPortionDeletion	Inserts a portion of the viral sequence with a deletion.

Junctions

Biologically, different types of junctions can occur where host and viral DNA meet including clean junctions, gaps and overlaps. These can be simulated in the synthetic viral integration data.

The type of junction (gap, whole or overlap) can be specified. If no junction type is specified, left and right junctions are randomly selected for each integration object. Integer values are assigned to each junction depending on its type. Clean junctions have a value of zero, gap junctions have a value between 1 and 10 inclusive and overlap junctions have a value between 1 and -10 inclusive. As overlaps are not allowed at both ends, either the left or right junction is assigned to one end and a clean junction assigned to the other. As gap junctions can occur at one or both ends a random selection is made to have a gap at the left, right, or both ends.

Details of the different types of junctions are outlined below,

Clean: Viral and host DNA meet without modification to the junction.

Gap: Simulates a gap between the host sequence and the integrated viral sequence. This could result from an error in DNA repair mechanisms. A stretch of DNA is randomly generated as the gap where the junction value is used as the length of the gap (Fig. 1).

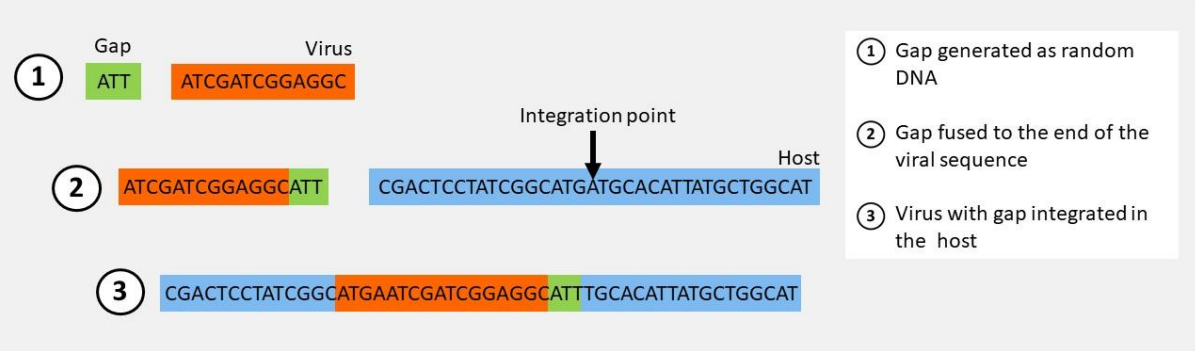

Fig 1: Insertion of a viral integration with a clean integration on the left end (junction = 0) and a gap on the right end (junction = 3).

Overlap: Simulates an overlap between the viral and host sequence. This could be caused by the viral sequence integrating in a location homologous to the host. The bases at the end of the

sequence with the junction are used for the overlap. The absolute value of the junction value is used as the length of the overlap. The integrated host sequence is then divided into left and right sequences at the assigned integration point. The left subsequence is scanned right to left for the overlapping bases. When a match is encountered, the site is checked for existing integrations. If an integration is found at this location, the remaining sequence is scanned for the overlapping region. This process is repeated for the right subsequence, instead scanning for the overlap from left to right. If no match is found in either subsequence the viral sequence cannot be inserted and the integration is aborted. If matches are found in both the left and right subsequences, the distance from the potential matches to the integration site is calculated. The match closest to the original integration site is used (Fig 2).

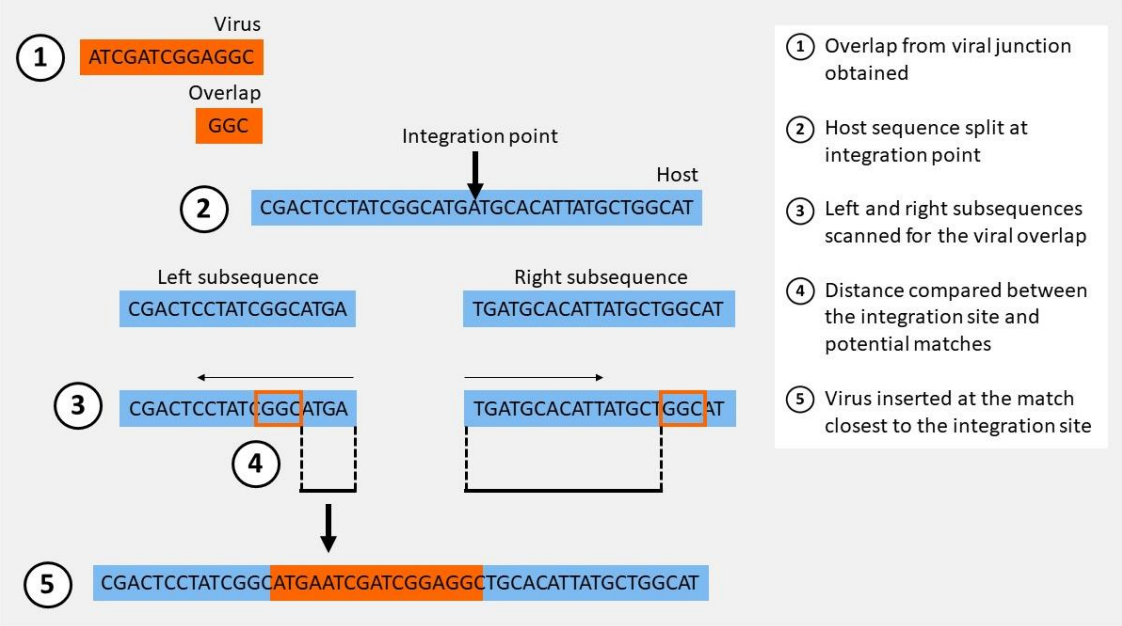

Fig 2: Insertion of an integration with a clean junction on the left (junction = 0) and an overlap junction on the right (junction = -3). Both the left and right subsequences contain matches with the overlap. The distance from the left match to the integration site is less than the distance from the right match to the integration site.

Episomes

AAV forms circular concatemers which persist in transduced cells. As integration of AAV is rare (Naso, et al., 2017), the amount of episomal AAV DNA in the cell nucleus is much greater than the amount of integrated AAV DNA. As episomes may also be sequenced, there is a possibility that sequencing reads derived from episome may be mistaken by the pipeline as integrations.

The option to add episomes to the output FASTA file is in included in *insert_virus.py* to assess the specificity of the pipeline for true integrations. Episomes are created using the Episomes class. Several different types of episomes can be included (Table 2). These utilise the viralChunk class. As episomes exist as circular plasmids of viral DNA, conversion to linear DNA is simulated by splitting the viral sequence in two at a random point and moving the first segment to the right of the second segment. This process occurs for all types of episomes.

Table 2: Types of episomes included in *insert_virus.py*

Type	**Description**
insertWhole	Episome made from whole viral sequence.
insertPortion	Episome made from a portion of the viral sequence.
insertWholeRearrange	Episome made from the whole viral sequence with a rearrangement.
insertWholeDeletion	Episome made from the whole viral sequence with a deletion.
insertPortionRearrange	Episome made from a portion of the viral sequence with a rearrangement.
insertPortionDeletion	Episome made form a portion of the viral sequence with a deletion.

Output

Three files are returned by *insert_virus.py.* The locations of these files must be parsed to the script. The first of these files, locs, is a text file updated with the details of each integration as they occur. These details include the chromosome, virus, hPos, number of fragments, orientation, whether it was rearranged or had a deletion, the breakpoints of any splittings, type of junctions, the start and stop positions of the viral chunk, whether the integration was inserted successfully and the sequence of the viral chunk. This file is useful for debugging if the inserted integrations appear different from the types of integrations selected and for finding details of specific integrations the pipeline may have been unable to detect. A separate .csv file, host_ints, contains the location of each integration in the host after all integrations have occurred. This is constructed using the Statistics class which adjusts the start and stop positions of each integration to find the position of each integration in the integrated host. This file does not consider bases included in gaps and

overlaps as part of the inserted virus. After NGS reads have been simulated, this file is used to determine which reads contain viral integrations.

Usage

Insert 300 random integrations

Python3 insert_virus.py --host your_host.fa --virus your_virus.fa --ints integrated_host.fa --locs details_on_integrations.txt --host_ints integration_locations_in_host.csv --int_num 300

Insert 100 integrations and 300 episomes

python3 ../intvi_pipeline/benchmarking/insert_virus.py --host test_host.fa --virus test_virus.fa --ints ints.fa --locs locs.txt --ints_host host_ints.csv --int_num 100 –epi_num 300

Insert 300 rearranged integrations with gaps

python3 ../intvi_pipeline/benchmarking/insert_virus.py --host test_host.fa --virus test_virus.fa --ints ints.fa --locs locs.txt --ints_host host_ints.csv --int_num 300 --int_type gap –junc_type gap

Insert 500 clean random integrations with a length of 240bp

python3 ../intvi_pipeline/benchmarking/insert_virus.py --host test_host.fa --virus test_virus.fa --ints ints.fa --locs locs.txt --ints_host host_ints.csv --int_num 300 --int_type clean --set_len 240

GENERATE READS

Paired end next-generation sequencing (NGS) reads are generated using ART Illumina (Huang, et al., 2012) by parsing the integrated host FASTA from *insert_virus.py*.

ART requires an average fragment size, standard deviation, and read length. Paired-end or single-end sequencing can also be selected. ART also has an option to set subsititution error-rate. Data with a low-error rate is more useful for detecting bugs in the pipeline as false negatives resulting from a pipeline bug can not be mistaken for reads with sequencing errors. Synthetic data with errors is more reflective of what results would be obtained from an experimental dataset.

ART returns a SAM file and FASTQ files of the simulated reads. These FASTQ files must be zipped before parsing to the pipeline.

read_integrations.py

The script, *read_integrations.py,* determines which of the generated NGS sequencing reads contain viral integrations. This is required to determine the effectiveness of the viral integration pipeline.

The SAM file containing the NGS reads and the host_ints file from *insert_virus.py* must be parsed. If only a subset of the reads in NGS file are of interest, a text file containing the Read IDs (QNAME) of the reads of interest can be parsed to reduce the amount of time required to run the script. This is useful for debugging as often this script can take a long time to run.

The simplesam package is used to process the SAM file and extract the QNAME, POS and SEQ for each of the reads. Using POS, the left most-most mapping position of the read and the length of the SEQ, the location of each read in the integrated host is determined. These positions are adjusted as the SAM file format uses a 1-based coordinate system whilst *insert_virus.py* uses a 0-based coordinate system. The location of each integrations in the integrated host and the hPos of each integration is obtained from the --host_ints file. Presence of viral DNA in each read is identified by comparing the start and stop positions (coordinates) of each of the reads with the coordinates of all the integrations. If the coordinates are found to overlap, the read must contain an integration. These overlaps may result in reads which are chimeric, contain short integrations or contain integrations at each end. Each type of read is given an abbreviation (Table 1). The type of read, amount of viral DNA (bp) in each read, location of each read in the integrated host and the hPos of any integrations in each read is assembled as a dataframe. This dataframe is used to create two output files. The first output file, --save, saves the entire dataframe as a .csv file and the second output file, --viral_reads, saves only rows of the dataframe corresponding to reads containing viral integrations. Whilst this may seem redundant, the save file is typically very large, often making it easier to work with the viral_reads file.

Table 1: Abbreviations used for different read types

Read type	**Abbreviation**
Viral	v
Host	h
Chimeric. Virus on left and host on right.	vh
Chimeric. Virus on right and host on left.	hv
Short integration read	sh
Split end read	sp

Compared to *insert_virus.py, read_intergrations.py* takes longer to run, especially if a high coverage is used for simulating NGS reads. The percentage completion is displayed as the program iterates through coordinates of the reads and integrations.

Two .csv files are produced by the script. The first saves details on all of the reads and the second saves details only on reads containing integrations. The location of both files, --all_reads, --viral_reads respectively, must be parsed to the script. Whilst this seems redundant, the –all_reads file is typically much larger than –viral_reads. Both files contain the read type, the amount of viral DNA in each read, the coordinates of each read and the hPos of any integration in the read.

Typically, I run *insert_virus.py,* ART and *read_integrations.py* as a single batch job. An example, createData_example.sh is included in the benchmarking directory.

Usage

Paired end

python3 read_integrations.py --sam your_sam.sam --host_ints integration_locations_in_host.csv –save all_reads.csv --viral_only viral_reads.csv --paired True

Single reads

python3 read_integrations.py --sam your_sam.sam --host_ints integration_locations_in_host.csv –save all_reads.csv --viral_only viral_reads.csv

Paired end sequencing filtering reads with a list of read IDs

python3 read_integrations.py --sam your_sam.sam --host_ints integration_locations_in_host.csv –save all_reads.csv --viral_only viral_reads.csv --paired True --filtered list_of_read_IDs.txt

pipeline_success.py

After using the viral integration pipeline to detect viral integrations in the simulated data, details of the reads predicted to contain integrations are returned. By comparing the Read IDs of the reads detected by the pipeline with the reads known to contain viral integrations we can assess the quality of the predictions made by the pipeline.

pipeline_success.py requires several files. These include the locs file from *insert_virus.py*, all_reads and viral_reads from *read_integrations.py* and the text file describing integrations detected by the pipeline. A directory, evaluate_pipeline_success, is created to store output files. As the format of the

Read IDs is different between the pipeline output and –all_reads, these strings are modified so they can be compared.

The viral integration pipeline can only detect integrations if there are at least 20 bp of viral DNA and 20bp of host DNA. The reads known to have integrations, viral_reads, are filtered to contain only reads which contain at least 20bp of viral and host DNA. Discordant read pairs whereby one read is entirely viral DNA and the other read contains a less than 20 bp of viral DNA and vice versa are not filtered.

Using the different files, four lists of Read IDs are created: reads containing integrations, reads which do not contain integrations, reads the pipeline predicted to contain integrations and reads the pipeline predicted to not contain integrations. Using these lists, the reads can be sorted into true positives, true negatives, false positives and false negatives. The number of entries in each of the lists are used to calculate metrics including accuracy, sensitivity, specificity and precision. The confusion matrix containing the number of true positive, true negative, false positive and false negatives is saved as a .csv file. The confusion matrices resulting from multiple experiments can easily be imported into a Jupyter notebook allowing visualisation of results. Lists of the false positive read IDs and false negative read IDs are also saved to text files. These lists are useful for debugging the pipeline.

A method is included to filter the integrations detected by the pipeline using different parameters in the pipeline output including edit distance, vector arrangement and translocation. This was not particularly useful for testing the pipeline as the pipeline had a low false positive rate. The results after filtering are saved as a bar graph. This uses a cairo backend.

Using the ints file, the number of integrations detectable from the pipeline results can be determined. Whilst the pipeline may not always be capable of detecting every read with an integration, if the coverage is high, a large portion of the integrations may still be captured. This is determined as the percentage of integrations detected.

Usage

python3 pipeline_success.py --pipeline pipeline_integrated_reads.txt --ints locs.txt --viral_reads viral_reads.csv --all_reads all_reads.csv --save location_to_create_directory

false_read_details.py

In the process of making modifications to the viral integration pipeline it was helpful to look at the CIGARS of false positive reads and false negative reads or false positive reads. Using the simplesam package, *false_read_details.py* assembles the CIGAR, sequence, secondary alignment (XA) and supplementary alignment (SA) from a BAM file for a list of read IDs. While this is usually efficient for obtaining details from the host alignment, as the alignment for the viral BAM is typically large, it often more helpful to manually grep the viral alignment for problem reads than to use this script.

USAGE

python3 --ID list_read_IDs.txt --bam your_bam.bam --save read_details.csv

REFERENCES

Huang, W.*, et al.* ART: a next-generation sequencing read simulator. *Bioinformatics* 2012;28(4):593-594.

Naso, M.F.*, et al.* Adeno-Associated Virus (AAV) as a Vector for Gene Therapy. *BioDrugs* 2017;31(4):317-334.