Post-pipeline integration site quantification

# Data Structure: Targeted bases

We define **targeted bases** as classes of sequencing reads that share each other the mapping location of the host-vector genome-junction when aligned to the host reference genome. In our set-up, any collection of sequencing reads can be partitioned in targeted bases, each of which is related and fully determined by a certain genomic coordinate.

In the following, with the term “targeted bases”, we'll refer to the genomic coordinates of the integration locus as well as to the related sequencing reads: hence a targeted base has also a sequence count, a set of shear sites and a set of random barcodes.

# Integration Sites computation

We group targeted bases each other according to a *proximity criterion* in a linear vision of the genome. We merge targeted bases in the same group as they were one, called **Integration Site (IS)**, whose genomic coordinates are chosen within the group according to a *strength of signal criterion*.

Thus an Integration Site, in this context, has to be seen as a targeted base itself, resulted from the consolidation of nearby targeted bases in one, in an attempt to trace out where the real integration event was supposed to take place, and gather related sequences, usually spread in surroundings due to PCR-artifacts, sequencing-error or mapping-issues. (cit. …?)

### About covered bases grouping

In a linear vision of the genome, starting from the *first* targeted base, we extend a window till *N* nucleotides beyond (default: 8). Such window may be even prematurely truncated, right after any targeted base encountered during its expansion, if the next one is farther then *D* nucleotides apart (default: 7) or if the chromosome ends. All the targeted bases in such a window are considered as member of the same group and they will be processed together to yield a single IS.

Then, from the next targeted base, a new window starts to expand, and this process is repeated till all data have been exhausted and all the targeted bases have been properly grouped. We called such groups **targeted base ensembles**, and usually in our set-up most of them are trivial (singleton).

### About IS Genomic Coordinates choice

In case of trivial ensembles there are no choices to perform, it’s just a matter of title, the targeted base becomes straightforward an IS. Conversely, if the ensemble is made up of more than one targeted base, we have to choose genomic coordinates of the resulting IS among them.

We perform this choice taking the coordinates of the targeted base with the highest sequence count: in case of ties, we take the first, *strand-wise*, still in a linear vision of the genome.

### About data merging

In case of trivial ensembles there is no need to merge anything, as said before it’s a matter of title, the targeted base becomes straightforward an IS. Conversely, if the ensemble is made up of more than one targeted base, after having gathered them under common genomic coordinates, related data need to be corrected and merged in light of this.

Since the new genomic coordinates are supposed to be the *true* ones, sequence lengths have to be recomputed. Having corrected shearsites this way, also the distinct random barcodes found for each of them have to be recomputed. The total sequence count for the newborn IS is simply the sum of the sequence counts inherited from source targeted bases. Indeed, in our workflow, we also keep track of the sequence count of each distinct (shearsite, random-barcode) couple for each IS.

# Random Barcode Edit-distance Filter

Due to sequencing error or PCR-artifacts, single or few nucleotide variations in random barcode sequences may occur. This issue leads to the overestimation of the distinct barcodes found in each IS, if taken as they are.

However, sequences sharing the same shearsite in an IS are usually many order of magnitude less than the distinct barcodes exploited for tagging (theoretically ). For this reason, is expected that such sequences, *in most of cases*, are ligated to random barcodes whose mutual edit-distances are *quite high*. This is naively our rationale, which exploits edit-distances under a certain threshold to spot adulterated barcodes.

### Edit Distance Definition

We implemented the definition of **Levenshtein**. Informally, Levenshtein distance is a string metric between couple of words, whose value is the minimum number of single-character edits (i.e. insertions, deletions or substitutions) required to change one word into the other.

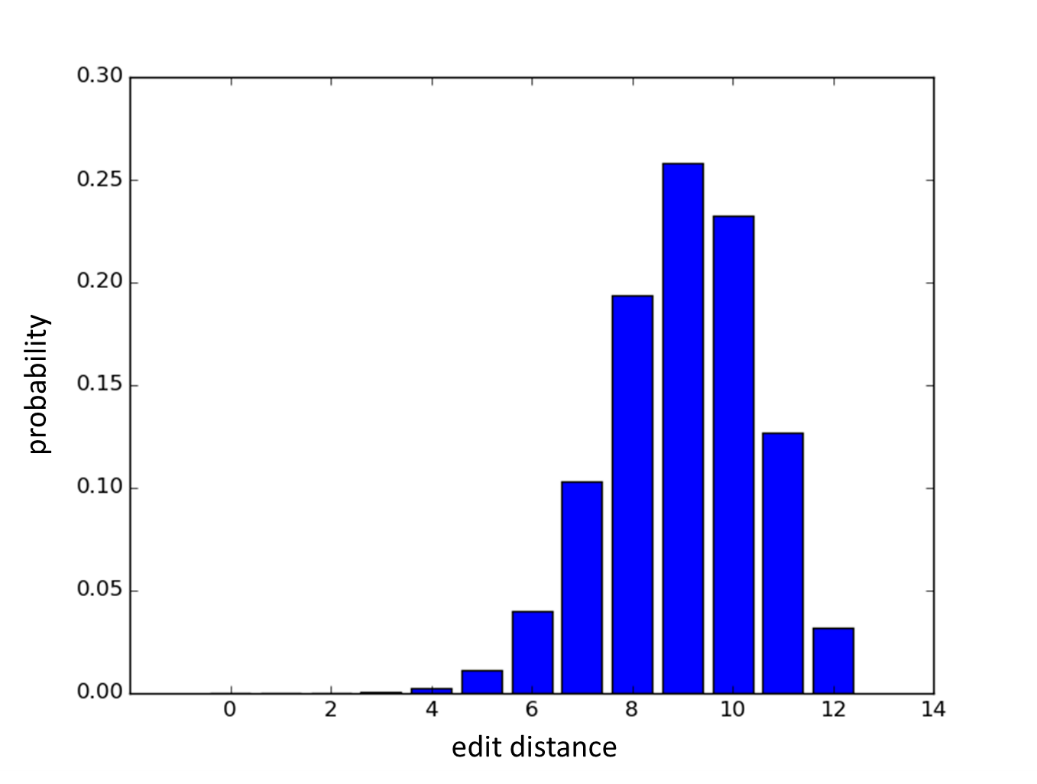
### Edit Distance threshold

We set the edit-distance threshold exploited to spot adulterated barcodes following a hybrid approach that starts off with theoretical expectations to find constraints and lastly relies on a piece of controlled data, as calibration, to get a number. Assessing results of the *algorithm* with the just found *setting*, upon our controlled dataset, was our validation strategy for *both*.

Theoretical expectations

Assume that all the random barcodes are available, homogeneously represented and equally likely to ligate to any DNA fragment. In other words, assume that the tagging is really random.

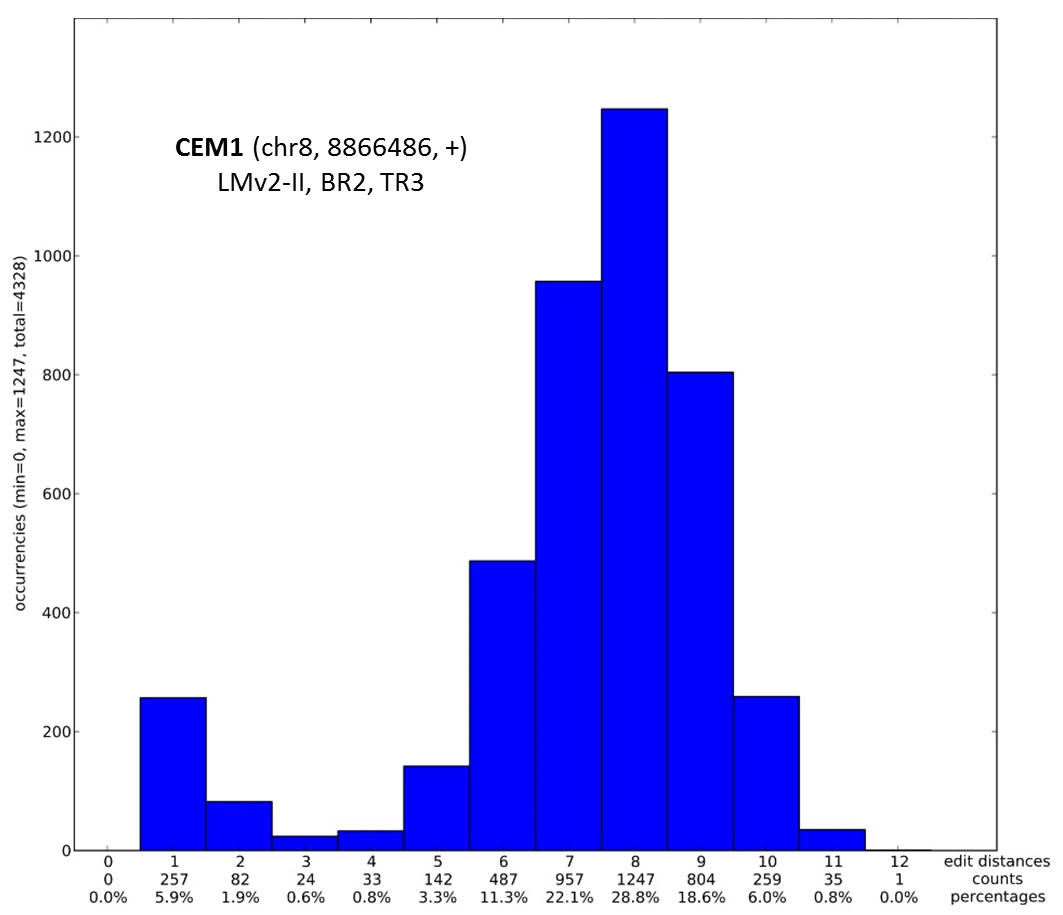
Further, assume that we are dealing with ideal data, not affected by any bias, inefficiency, contamination, sequencing or alignment error and so on.

Under these hypotheses, any sample of random-barcodes from such data should yield a binomial distribution of mutual edit distances *all-VS-all*, as shown in the picture below.

We got two important observations from this distribution:

1. Since *Mean ≈ Median ≈ Mode = 9* and *2 x Variance ≈ 3*, it is expected that *~95%* of edit distances found in any sample of such ideal data is in a range between 6 and 12. So, theoretically, adulterated barcodes in real data could be spotted by an edit distance of 5 or less, with an error rate of <*~*5%.
2. Starting from the mode and moving left towards zero, we observe the strictly decreasing trend of the probability shape, that drops down to for (edit distance of 3). Further, the number of expected events as much extreme or more is still negligible , so we can safely claim that all the random barcode couples yielding an edit distance of 3 or less contain an artifact.

Calibration on data

We computed the histogram of edit distance occurrences for all the 9 replicates and triplicates of the LMv2-II CEM1 data, dilution L. This choice was taken because here we have the strongest signal available for a single integration site (CEM1 at 70%). Replicates and triplicates are consistent, so we take just one as an example, here below.

is the threshold value starting from whom the decreasing trend of the shape stops, and start to raise. Also, the observed frequency is already higher than the expected one.

For these reasons we selected **3** as edit-distance threshold: all the random barcode couples under this value, are inspected and filtered by the algorithm presented in the following.

### Filtering algorithm

Edit distance is a relation that holds among *couples* of barcodes and returns a number. A threshold of 3 may allow to spot couples supposed to host at least one adulterated sequence. We don’t know which of them is the original one, maybe neither of the two, if both of them were artifacts generated from a third sequence not evaluated yet by any sequential algorithm.

To overcome this hurdle, we exploited the sequence count of the barcodes to **set an order to be followed while comparing couples**, as well as to set a rule that allows to assess, in spotted couples, which barcode is the original one and discard the counterpart. The rationale is**: the higher is the sequence count for a given barcode, the more likely it could be a non-adulterated one**, because of the intrinsic randomness of errors; it’s not very likely that independent errors generates many identical outcomes, giving rise to an adulterated sequence with a high count.

Relying upon this idea, we developed a sequential algorithm that can run just one, quite fast, without the need to enumerate all the possible scenarios, and neither roll back to reconsider previous choices in light of facts discovered during the computation.

How it works

The filtering protocol run over each integration site in each sample, independently, because each sample underwent an independent barcode ligation.

Indeed, sequences related to each IS of each sample are grouped by shearsite, because is not always true that the same shearsite is shared only among sequences from the same parental fragment – actually this is the reason why random barcoding was introduced – but for sure distinct shearsites comes from distinct ones.

Such subgroups of sequences – sharing the same shearsite, in the same IS, from the same sample – are the fundamental units processed by the filtering algorithm.

Main steps

1. Sequences are sorted by sequence count, from the highest to the lowest. In case of ties, we formerly thought to shuffle them randomly to reduce biases, but lastly we opted for sorting lexicographically to guarantee deterministic results.
2. The first random barcode (with the highest sequence count, supposed to be *true*) is compared to the other N-1 barcodes: if a comparison returns an edit distance under the threshold value (default: 3), the counterpart barcode is labelled as “to discard” and won’t be kept in final data. It’s always the counterpart barcode to be called into question, by construction, because of the lower sequence count, that makes it more likely to be the adulterated one than the other.
3. The second random barcode (with the second-higher sequence count) is compared to the other N-2 barcodes left, and the same logics as in point (2) are applied. And so on, till the end of the barcodes.

Note

The filtering algorithm has no impact on the number of ISs, it just may varies the number of distinct couples (shearsite,barcode) per ISs, and hence quantifications leveraging on this.

# Quantification Approaches

We quantify Integration Site data in 5 different fashions. Quantifications are expressed in matricial form, where each row represents an IS whose quantification is demultiplexed in columns according to *sample IDs* found in data.

### 1 – seqCount Method

This method quantifies integration sites simply counting the number of related sequencing reads. Since barcode data are not exploited here, the filtering algorithm is not applied.

In matricial form, a cell (i,j) shows the number of sequencing reads found in sample j for the integration site i.

This method is usually biased by PCR-like amplification techniques. (cit. …?)

### 2 – ShsCount Method

This method quantifies integration sites counting the number of distinct shearsites found in related sequencing reads. Since barcode data are not exploited here, the filtering algorithm is not applied.

In matricial form, a cell (i,j) shows the number of distinct shearsites observed in sequencing reads from sample j for the integration site i.

This method is prone to “flatten” data, underestimating major IS (saturation of number of sonicants) and overestimating minor ones (each observed IS has at least one shearsite). (cit. Berry)

### 3 – barcodeCount Method

This method quantifies integration sites counting the number of distinct random barcodes found in related sequencing reads.

In matricial form, a cell (i,j) shows the number of distinct barcodes observed in sequencing reads from sample j for the integration site i.

This method may overestimate ISs randomly, since it is sensitive to single nucleotide errors on barcode sequences. A filter for adulterated sequences is required. (cit. Firouzi)

### 4 – cellCount Method

This method refines the number (3). It quantifies integration sites counting the number of distinct random barcodes *after grouping them by shearsite*. In other words, it counts the number of distinct couples (shearsite,barcode): if the same barcode is found attached to distinct shearsites, it counts twice.

It actually exploits barcode data to *demultiplex shearsites at the ‘genome granularity’*: each (shearsite,barcode) couple is supposed to be univocally linked to a genome of a cell that harboured that integration.

In this sense, in matricial form, a cell (i,j) shows the number of cell in sample j that harboured integration site i. (cit. Firouzi)

### 5 – fragmentEstimate Method

This method refines the number (2). Here, the enumeration of distinct shearsites is taken as input by a sophisticated statistical framework implemented in R (cit. Berry), that aims to estimate the number of parental fragment that gave rise such a shearsite’s scenario.

It provides excellent quantification, but they are still *maximum likelihood estimates*, instead of an empirical count that our piece of data allows. Also, it may happen that the model doesn’t fit data so well as well as that the algorithm take its time to converge.

# Implementation

The here presented workflow, from arranging data in targeted bases till quantification matrixes, was implemented as a python script (cit. python).

### Software Requirements

The script was designed to run in **Linux** environment, under **python 2.7** and requires **pandas** library (cit. pandas).

Other *soft requirements* are:

1. **editdistance** library (required only by the filtering algorithm part, the script can also run without it) (cit. editdistance)
2. **rpy2** (python interface to R); **R** installed and globally callable, equipped by **sonicLength** package (required only to get the ‘fragmentEstimate’ quantification matrix, the script can also run without them) (cit. R, rpy2, sonicLenght\_package)

### Avalability and Installation

Repository: <https://bitbucket.org/tigetbioinformatics/shearsites_analysis/branch/RandomBC>

The software is portable, fully contained in **matrix\_release** folder. Any other folder has to be ignored. At the time when this document was written, the latest commit was **ecd1361**.

### Expected input data

In brief, this software takes as input the output of VISPA2.

At the time when this document was written, “VISPA2” meant **38580ee** commit of <https://bitbucket.org/andreacalabria/isatk/branch/v3>.

**ONLY PAIRED-END READS ARE PROCESSED**, due to the need of shersite data. The criterion exploited to spot and discard not-properly-paired reads is “*mate chromosome is NULL*”.

### Launch

To launch the full workflow as presented here, with default values, open a terminal in **Matrix\_release** folder (or wherever, properly changing the script file path) and type:

$ python \_matrix\_MAIN.py ${NGSWORKINGPATH},${DISEASE},${PATIENT}

--dataset\_ID *user-choice-string-to-label-data*

The *dataset tuple* ${NGSWORKINGPATH},${DISEASE},${PATIENT} can be further specified to take just some pools

[…] ${NGSWORKINGPATH},${DISEASE},${PATIENT},${POOLNAME1}

Or

[…] ${NGSWORKINGPATH},${DISEASE},${PATIENT},${POOLNAME1},${POOLNAME2}

And so on.

Many dataset tuples are allowed simultaneously, just adding them next, separated with a space.

This feature was introduced because integration site computation is made by default over the input *as a whole*, unlike all the other operations that are *per-sample*. However, also integration sites can be computed *per-sample,* through--ISs\_per\_sample option. This way, two distinct launches with two datasets or one single launch with both, would produce consistent results.

Here we indicated strings as content of VISPA2 variables because they are actually congruent.

### Output

According to the features of the whole input data, and to your system configuration, this program yields as output up to five quantification matrix files:

1) *user-choice-string-to-label-data* \_seqCount\_matrix.tsv

2) *user-choice-string-to-label-data* \_ShsCount\_matrix.tsv

3) *user-choice-string-to-label-data* \_barcodeCount\_matrix.tsv

4) *user-choice-string-to-label-data* \_cellCount\_matrix.tsv

5) *user-choice-string-to-label-data* \_fragmentEstimate\_matrix.tsv

File contents should be obvious, in light of previous “Quantification Approaches” section; *user-choice-string-to-label-data* is the --dataset\_ID’s argument, as shown in the previous subsection.

File format is plain text, utf-8, tab-separated. Empty cells stand for N/A’s.

By default, such files are created in the current working directory. -o / --out\_dir\_path allows to specify the full path of any directory (existing or to create) as alternative destination.

In the same location where the user chooses to write quantification matrix files, a subfolder named 'MatrixesTotal' is expected: if not existing, it will be created. This subfolder hosts the total-counterpart of quantification matrix files, i.e. a condensed-in-one-column version of related quantification matrix files. Such files have the same names of their counterpart except for *'total\_*' prefix. The only column inside them is labelled, again, as *user-choice-string-to-label-data.*

### Other options

See help

$ python \_matrix\_MAIN.py --help

# References (DRAFT)

(cit. …?): citazione necessaria, facile da trovare, non penso a nessun paper in particolare.

(cit. Berry): Berry CC, Gillet NA, Melamed A, Gormley N, Bangham CRM, Bushman FD. Estimating abundances of retroviral insertion sites from DNA fragment length data. Bioinformatics. 2012;28(6):755-762. doi:10.1093/bioinformatics/bts004.

(cit. Firouzi): Firouzi S, López Y, Suzuki Y, et al. Development and validation of a new high-throughput method to investigate the clonality of HTLV-1-infected cells based on provirus integration sites. Genome Medicine. 2014;6(6):46. doi:10.1186/gm568.

(cit. python): Python Software Foundation. Python Language Reference, version 2.7. Available online at <http://www.python.org>

(cit. pandas): Wes McKinney. Data Structures for Statistical Computing in Python, Proceedings of the 9th Python in Science Conference, 51-56 (2010)

(cit. editdistance): <http://pypi.python.org/pypi/editdistance>. The algorithm used in this library is proposed by Heikki Hyyrö, “Explaining and extending the bit-parallel approximate string matching algorithm of Myers”, (2001).

(cit. R, rpy2, sonicLenght\_package):

* R: R Development Core Team (2011), R: A Language and Environment for Statistical Computing. Vienna, Austria: the R Foundation for Statistical Computing. ISBN: 3-900051-07-0. Available online at <http://www.R-project.org/>
* rpy2: Interface to R running embedded in a Python process. Available at <http://rpy2.bitbucket.org/>
* sonicLenght\_package: Estimating Abundance of Clones from DNA fragmentation data. Available online at <http://CRAN.R-project.org/package=sonicLength>.