IGIA Documentation

Release 1.0

Dehe Wang

Jan 25, 2019

CONTENTS:

CHAPTER

ONE

OVERVIEW

1.1 Download IGIA

```
git clone https://github.com/zhouyulab/igia.git path/to/igia
```

1.2 Prepare the virtual environment

IGIA is implemented in Python, and depends on several packages. With the installation and activation of virtual environment (with Conda) as shown below, you can ensure that the tools run properly.

1.2.1 Step 1: Download Miniconda3

```
wget https://repo.continuum.io/miniconda/Miniconda3-latest-Linux-x86_64.
sh
```

```
bash Miniconda3-latest-Linux-x86_64.sh
```

1.2.2 Step 2: Create a new Python3 environment

```
conda create -n igia python=3.5
```

1.3 Install IGIA

```
cd path/to/igia
```

```
source activate igia
```

```
pip install -r requirements.txt
```

```
python setup.py install
```

With the setup of the environment, the following packages are installed automatically.

- [python](https://www.python.org/) (v3.5.3)
- [pysam](https://pysam.readthedocs.io/) (v0.10)
- [numpy](https://www.numpy.org/) (v1.11.3)
- [scipy](https://www.scipy.org/) (v0.18.1)

- [networkx](https://networkx.github.io/) (v1.11)
- [deepTools](https://deeptools.readthedocs.io/) (v2.5.1)
- [bx-python](https://pypi.org/project/bx-python/) (v0.7.3)
- [pybedtools](https://daler.github.io/pybedtools/) (v0.7.10)
- [pyBigWig](https://pypi.org/project/pyBigWig/) (v0.3.4)
- [mpi4py](https://pypi.org/project/mpi4py/) (v2.0.0)

1.4 Execute IGIA

To run IGIA with single-threaded mode, you can execute:

```
source activate igia

igia --tgs tgs1.bam --tss tss.csv --tes tes.csv --ngs ngs1.bam ngs2.
bam -o igia_res
```

To run IGIA with MPI mode in a cluster, you must first ensure that Openmpi/Mpich is installed and already configured in the cluster. Then you can execute:

```
source activate igia

mpirun -genv I_MPI_DEVICE ssm -n 8 igiampi --tgs tgs1.bam --tss tss.
csv --tes tes.csv --ngs ngs1.bam ngs2.bam -o igia_res
```

1.5 Execute IGIA with test data in the package

If you have successfully installed IGIA, you can use the following command to run IGIA on test data.

```
cd /path/to/igia/test

bash example.sh
```

This demo run on example data will execute for a few minutes, and the results from IGIA will be generated in /path/to/igia/test/igia_res.

The results contain iso*.bed12 files, a set of assembled transcripts in BED12 format, and 4 *.bed6 files for different genomic elements identified.

For IGIA assembled transcripts, isoF and isoA are the most reliable annotations. For details, please refer IGIA manuscript.

CHAPTER

TWO

USAGE

Integrative Gene Isoform Assembler

```
usage: igia [-h] [--version] [-v] [-vv] -o out_dir --ngs
            [NGS.bam [NGS.bam ...]] --tgs [TGS.bam [TGS.bam ...]]
            [--tss tss.tsv] [--tes tes.tsv] [--ann NGS_ann.bed12]
            [--cfm-ann comfirmed_ann.bed12] [-s chrom.size] [-g genome.fa]
            [-r rule] [--pir pir] [--dtxs dtxs] [--time-out time_out]
            [--paraclu-path paraclu_path]
```

2.1 Named Arguments

--version	show program's version number and exit
-v, --verbose	set loglevel to INFO
-vv, --very-verbose	set loglevel to DEBUG

2.2 Base

-o, --output	Output folder for IGIA assembles
--ngs	Next generation RNA sequencing data in BAM format
--tgs	Long reads RNA sequencing data in BAM format

2.3 External

--tss	TSS information (TAB separator file, ChromtSitetStrand)
	Default: ""
--tes	TES information (TAB separator file, ChromtSitetStrand)
	Default: ""
--ann	Bed12 format NGS based annotation, needs –size for chromsize
	Default: ""
--cfm-ann	Bed12 format comfirmed annotation, needs –size for chromsize
	Default: ""

-s, --size	Chrom size file generated by twoBitInfo Default: “”
-g, --genome	Genome sequence file in FASTA format

2.4 Options

-r	Possible choices: 1++,1–,2+-,2-+, 1+-,1-+,2++,2–, single_end NGS library type Default: “single_end”
--pir	PIR cutoff for intron retention [default=0.5] Default: 0.5
--dtxs	Distance cutoff between two different TSS/TES [default=500] Default: 500
--time-out	TimeOut
--paraclu-path	Path to paraclu (for detected TSS and TES from TGS data)

CHAPTER

THREE

LIBRARY REFERENCE

3.1 utils

class igia.utils.**SeqFile**(*filename*, *file_type*)

Bam format file

chrom2size()

Compute chrom size.

Returns {chrom: size}

Return type dict

chromsize()

Compute chrom size.

Returns (chrom, size)

Return type list

count(*chrom*, *start*, *end*)

Compute read number in ival.

fetch_reads_in_ival(*ival*, *skip_boundary_span=True*)

Fetch reads in interval.

filter_clean_reads(*iter*)

Filter read by rules.

genomesize()

Compute genome size.

mapped_number()

Compute read number.

poisbg(*binsize*, *alpha=0.01*)

Compute NGS signal background by poisson distribution.

Parameters

- **binsize** (*int*) – Window size
- **alpha** (*float*) – Confidence coefficient

Returns Background cutoff

Return type float

pretreat(*read*)

Read pretreatment

Parameters **read** (*AlignedSegment*) – read to pretreatment

Returns read

Return type AlignedSegment

readlen(*top=100*)
Compute read length.

smart_fetch(*chrom*, *start*, *end*)
Fetch reads in ival.

class igia.utils.**GenomeFile**(*file_dir*)
FASTA format file

find_sequence(*chrom*, *start*, *end*)
Fetch sequence from genome

Parameters

- **chrom** (*str*) – Chromosome to fetch
- **start** (*str*) – Start position
- **end** (*str*) – End position

Returns Sequence

Return type str

class igia.utils.**Coverage**(*ival*)
Wig like coverage format.

build(*ngs_list*)
Make coverage by NGS data.

slice(*ival*)
Fetch subregion coverage.

class igia.utils.**Interval**(*chrom*, *start*, *end*, *strand*)
Interval information and operation.

build_cov(*ngs_list*)
Make coverage by NGS data.

compute_fpkm(*readlen_list*, *readnum_list*, *alpha=0.01*)
Compute FPKM from coverage

classmethod cov2fpkm(*cnt*, *ivlen*, *readnum*, *alpha*)
Formular from GRIT

inherit_cov_from(*ival*)
Slice coverage signal from another interval

slice_cov(*ival*)
Fetch subregion coverage.

class igia.utils.**Bed12**(*line*)
Bed12 format

find_intron()
Find intron from the record

iterblock()
Iterate exon

rel2abs(*rel_pos*)
Transform relative mRNA position to absolute genomic position

write(*f*)
Write record to file

class igia.utils.**AlignReadMethod**
Abstract class for a read

classmethod fetch_seq_by_ref_loc(*read*, *start*, *end*)
Fetch sequence for a read

Parameters

- **read** (*AlignedSegment*) – Read
- **start** (*int*) – Start position
- **end** (*int*) – End position

Returns RNA-seq sequence

Return type str

classmethod has_intron(*read*, *start*, *end*)
Determine if there is an intron inside a region

Parameters

- **read** (*AlignedSegment*) – Read to scanning
- **start** (*int*) – Start position
- **end** (*int*) – End position

Returns has intron

Return type bool

classmethod ref_loc2query_loc(*read*, *start*, *end*)
Transfer the reference position to query position

Parameters

- **read** (*AlignedSegment*) – Query read
- **start** (*int*) – Reference start site
- **end** (*int*) – Reference end site

Returns tuple: query_start, query_end

igia.utils.**load_seqinfo**(*ngs_bams*)
NGS_FILE_INFO_list: (file_name, read_num, read_len)

igia.utils.**poiscut**(*totread*, *genomesize*, *binsize*, *alpha=0.01*)
Compute the cutoff of count for given binsize with totread number

igia.utils.**load_txs**(*file*)
Load tsv format TXS annotation :param file: TXS file :type file: str

Returns TXS list

Return type list

igia.utils.**make_read**(*ref_id*, *name*, *start*, *cigar*)
Build sam format read by position

igia.utils.**bed2bam**(*f_bed*, *f_chromsize*, *outdir*)
Transfer Bed format file to Bam format file

Parameters

- **f_bed** (*str*) – Bed format file
- **f_chromsize** (*str*) – Chrom size file
- **outdir** (*str*) – Outout folder

Returns Bam format file

Return type str

igia.utils.**iterline**(*file*)
Read file with cache

igia.utils.**load_ann**(*f_ann*, *size*, *out_dir*, *type*)
Load transcript annotation

Parameters

- **f_ann** (*str*) – Bed12 format annotation file
- **size** (*str*) – Chrom size file
- **out_dir** (*str*) – Output folder to storage bam format file
- **type** (*str*) – Annotation file type

Returns Annotation sequence object

Return type *SeqFile*

3.2 coverage

class igia.coverage.**CountReadsPerBinWithIntron**(*bamFilesList*, *binLength=50*, *numberOfSamples=None*, *numberOfProcessors=1*, *verbose=False*, *region=None*, *bedFile=None*, *extendReads=False*, *blackListFileName=None*, *minMappingQuality=None*, *ignoreDuplicates=False*, *chrsToSkip=[]*, *stepSize=None*, *center_read=False*, *samFlag_include=None*, *samFlag_exclude=None*, *zerosToNans=False*, *smoothLength=0*, *minFragmentLength=0*, *maxFragmentLength=0*, *out_file_for_raw_data=None*)

count_reads_in_region_with_intron(*chrom*, *start*, *end*, *bed_regions_list=None*)
Rewrite deeptools.CountReadsPerBin.count_reads_in_region

Parameters

- **chrom** (*str*) – Chrom
- **start** (*int*) – Start position
- **end** (*int*) – End position

- **bed_regions_list** (*list*) – List of bed region

Returns subnum_reads_per_bin, file_name

Return type tuple

get_coverage_of_region_with_intron(*bamHandle*, *chrom*, *regions*, *fragmentFromRead_func=None*)

Rewrite deeptools.CountReadsPerBin.get_coverage_of_region

Parameters

- **bamHandle** (*AlignmentFile*) – Bam object
- **chrom** (*str*) – Chrom
- **regions** (*list*) – List of block
- **fragmentFromRead_func** (*function*) – Function to get fragment from read

Returns coverages

Return type float

3.3 linkage

class igia.linkage.**Linkage**

List of blocks by chrom

add_chr_linkage(*chrom*, *region_list*)

Merge the information from another linkage for certain chrom :param chrom: Chrom to merge :type chrom: str :param region_list: List of block :type region_list: list

add_linkage(*linkage*)

Merge the information from another linkage :param linkage: Another linkage :type linkage: Linkage

getregions(*chrom*)

Find linkage for a certain chrom.

Parameters **chrom** (*str*) – Chrom

Returns List of block region

Return type list

iterlinkage()

Iter linkage in whole genome

igia.linkage.**find_linkage_worker**(*chrom_size*, *seq_obj*)

Find RNA linkage from NGS and TGS data for one BAM file

igia.linkage.**find_linkage**(*bamlist*)

Find RNA linkage from NGS and TGS data

3.4 element

class igia.element.**GeneLinkage**(*chrom*, *start*, *end*, *ngs_bam_list*, *tgs_bam_list*, *ann=None*)

Linkage of read cluster.

cluster2gene(*cluster*, *f_genome=None*)

Convert Cluster object to Gene object

Parameters

- **cluster** (*list*) – Reads cluster
- **f_genome** (*str*) – Genome file

Returns Gene object

Return type *Gene*

fetch_reads()
Fetch reads in this linkage

Returns List of reads

Return type list

filter_txs(*txs_list*)
Select txs in the linkage :param txs_list: TXS list to filter :type txs_list: list

Returns List of TXS

Return type list

find_clusters()
Compute gene cluster

split2gene(*f_genome=None*)
Split GeneLinkage into Gene cluster.

Parameters **f_genome** (*str*) – Genome file

Returns List of gene

Return type list

class igia.element.**GeneLinkageFinder**
Abstract class for gene linkage

classmethod build_exon_overlap_cluster(*reads*, *overlap_ratio=0.5*)
Build exon overlap cluster :param reads: List of reads :type reads: list :param overlap_ratio: Overlap ratio to label two read in one cluster :type overlap_ratio: float

Returns List of reads cluster

Return type list

classmethod build_tgs_cluster(*reads*, *bam_list*, *chrom*, *overlap_ratio=0.5*, *txs_diff=400*, *strand_fraction=0.5*, *intron_cutoff=0.1*, *cov_cutoff=None*)
Use strand filtered tgs reads to build exonic overlapped gene cluster.

classmethod compute_read_overlap_ratio(*read1*, *read2*, *strand=True*)
Compute overlap ratio between two reads. :param read1: Read1 :type read1: AlignedSegment :param read2: Read2 :type read2: AlignedSegment :param strand: strand specific :type strand: bool

Returns (overlap_len/read1, overlap_len/read2)

Return type tuple

classmethod filter_cluster_by_strand(*cluster_list*, *strand_fraction=0.5*, *intron_cutoff=0.1*, *exon_cutoff=0.5*)
Clusters sharing same introns are considered as one gene

classmethod filter_nonspliced_cluster_by_cov(*cluster*, *bam_list*, *chrom*, *cov_cutoff*, *min_tgs_num=2*)
Non-spliced cluster should have enough FPKM

classmethod find_intron(*read*)
Find intron in a read :param read: Read :type read: AlignedSegment

Returns List of intron

Return type list

classmethod find_intron_in_cluster(*cluster*)
Find all intron in a cluster

Parameters **cluster** (*list*) – Reads cluster

Returns Set of intron

Return type set

classmethod merge_clusters(*clusters*)
Merge clusters into reads

classmethod split_cluster_by_overlap(*cluster*)
Split cluster by if overlap statement

class igia.element.**Gene**(*chrom*, *start*, *end*, *strand*, *tgs_read_list*, *ngs_bam_list*, *ann_read_list=None*, *f_genome=None*)
Linkage of gene cluster.

build_cov()
Create wiggle like signal coverage from bam files

identify_element(*ext_tss_site_list*, *ext_tes_site_list*, *paraclu_path=None*)
Identify transcript elements by TGS and NGS data.

identify_internal_exon()
Identify internal exon glue code.

identify_intron()
Identify intron glue code.

identify_tes_exon()
Identify tes exon glue code.

identify_tes_site(*ext_tes_site_list*)
Identify tes site glue code.

identify_tss_exon()
Identify tss exon glue code.

identify_tss_site(*ext_tss_site_list*)
Identify tss site glue code.

write_element2bed6(*f_intron*, *f_internal_exon*, *f_tss_exon*, *f_tes_exon*, *gene_name*)
Write elements into file with Bed6 format

class igia.element.**Exon**(*chrom*, *start*, *end*, *strand*)
A Structure to store exon data.

set_tes_exon(*is_tes_exon*)
Label this exon as TES exon or not

set_tss_exon(*is_tss_exon*)
Label this exon as TSS exon or not

class igia.element.**ElementDiscover**
General method in identify transcript elements.

classmethod adjust_intron_position(*tgs_read*, *intron*)
Adjust intron position if possible

classmethod blocks2cigar(*blocks*)
Produce cigar tuple by exon blocks

Parameters **blocks** (*list*) – list of exon block

Returns cigar tuple

Return type tuple

classmethod detect_txs_by_tgs(*txs_list*, *paraclu_path*, *maxlen=20*, *minDensityRise=4*, *minReads=5*)
Detect txs by paraclu

classmethod enumerate_exon(*intron_list*)
Enumerate possible internal exon by intron.

classmethod fetch_neighbor_seq(*start*, *end*, *ival*, *genome*)
Extraction sequence near a interval

classmethod fetch_splice_site(*intron*, *genome*)
Extraction splice site

classmethod find_txs_by_tgs(*tgs_list*, *txs_type*, *paraclu_path*)
Detect TXS by TGS data

Parameters

- **tgs_list** (*list*) – List of TGS reads
- **txs_type** (*str*) – TXS type in [tss, tes]
- **paraclu_path** (*str*) – Path to paraclu

Returns List of TXS

Return type list

classmethod fix_mapping_error(*ngs_intron_set*, *tgs_intron_set*, *genome*, *tgs_reads*, *ngs_bam_list*)
Unified NGS and TGS intron and adjust tgs reads' intron position

classmethod has_gap(*exon*)
Judge if a region have too many gap to by a exon.

classmethod identify_internal_exon(*intron_list*, *gene_ival*)
Identify internal exon by intron.

classmethod identify_tes_exon(*tss_site_list*, *tes_site_list*, *intron_list*, *gene_ival*)
Predict TES exon.

classmethod identify_tss_exon(*tss_site_list*, *tes_site_list*, *intron_list*, *gene_ival*)
Predict TSS exon.

classmethod ss2pri(*ss*)
Compute splice site priority

class igia.element.**NgsElementDiscover**
Method class to identify transcript elements by ngs data.

classmethod identify_intron(*gene_ival*, *ngs_bams*)
Predict intron by NGS data.

class igia.element.**TgsElementDiscover**
Method class to identify transcript elements by tgs data.

classmethod identify_intron(*gene_ival*, *tgs_read_list*)
Predict intron by TGS data.

class igia.element.**Intron**(*chrom*, *start*, *end*, *strand*, *spliced_readnum=0*)
A Structure to store intron data

is_spliced(*gene_ival*, *pircutoff=0.3*)
Compute if this intron is almost spliced

Parameters

- **gene_ival** (Interval) – Gene interval
- **pircutoff** – PIR cutoff

Returns bool

set_spliced(*spliced*)
Set splice information

set_spliced_readnum(*spliced_readnum*)
Set spliced read number

class igia.element.**JunctionGraph**(*ival*)
Use junction reads to build junction graph to identify intron.

class igia.element.**JunctionGraphNgs**(*ival*)
Use ngs junction reads to build junction graph to identify intron.

identify_intron(*bam_list*)
Reads to intron.

class igia.element.**JunctionGraphTgs**(*ival*)
Use tgs reads to build junction graph to identify intron.

identify_intron(*tgs_read_list*)
Reads to intron.

igia.element.**identify_element**(*chrom*, *start*, *end*, *ngs_bam_list*, *tgs_bam_list*, *ext_tss_site_list*, *ext_tes_site_list*, *ann=None*, *f_genome=None*, *paraclu_path=None*)
Identify transcript elements by TGS and NGS data.

Parameters

- **chrom** (*str*) – Chrom
- **start** (*int*) – Start position
- **end** (*int*) – End position
- **ngs_bam_list** (*list*) – List of NGS bam files
- **tgs_bam_list** (*list*) – List of TGS bam files
- **ext_tss_site_list** (*list*) – List of annotated TSS site (chrom, loc, strand)
- **ext_tes_site_list** (*list*) – List of annotated TES site (chrom, loc, strand)
- **ann** (SeqFile) – Annotation object
- **f_genome** (*str*) – Genome file
- **paraclu_path** (*str*) – Path of paraclu

Returns List of gene object

Return type list

3.5 transcript

class igia.transcript.**Segment**(*chrom*, *start*, *end*, *strand*)
Segment in identify transcript.

set_spliced_seg(*is_spliced=False*)
Label the segment as spliced segment or not

set_tes_seg(*is_tes=False*)
Label the segment as TES segment or not

set_tss_seg(*is_tss=False*)
Label the segment as TSS segment or not

class igia.transcript.**Isoform**(*segment_array*, *trans_ival*)
Isoform record

iso2bed12(*seglist*, *name*)
Transfer this isoform to Bed12 format string :param seglist: List of segment :type seglist: list :param name: Isoform name :type name: str

Returns Bed12 format string

Return type str

set_label(*label*)
Set label

set_tag(*tss_indxs*, *tes_indxs*, *label*)
Set tag of this isoform if this isoform is full-length isoform

Parameters

- **tss_indxs** (*list*) – List of TSS boundary index
- **tes_indxs** (*list*) – List of TES boundary index
- **label** (*str*) – tag to set

write2bed12(*seglist*, *name*, *f*)
Write this isoform to a file with Bed12 format

class igia.transcript.**TransAssembler**(*gene*, *ann=None*)
Identify transcript.

build_compatible_matrix()
Build the matrix that if two isoform are compatible

build_overlap_matrix()
Build the matrix that if two isoforms are overlapped on exons

build_subpath_matrix()
Build the matrix [i, j] that if isoform i contains isoform j

cluster_iso()
Compute the isoform cluster

get_isonum()
Compute full-length isoform number in this assembler

identify_isoform()
Engine for assembly

init_assembly()
Prepare isoform information for assebmly

init_segiso()
Init isoform

intron_path(*start_seg_indx*, *end_seg_indx*)
Build the intron path

make_seg_indx()
Compute the boundary of functional elements

make_segment()
Build segment list.

rescude_isoforms(*invalid_iso_list*)
Rescude the isoforms with errors

seglink_cnt
Count reads number supported the given segment pair

write2bed12(*cluster_name*, *f_isoF*, *f_isoA*, *f_isoR*, *f_isoM*, *f_isoC*, *f_isoP*)
Write all isoforms in this gene cluster to a file with Bed12 format

write_iso2bed12(*iso_type*, *gene_name*, *isolist*, *f*)
Write isoforms in same type to a file with Bed12 format

class igia.transcript.**TgsFilterRule**
TGS read filter rules.

classmethod filter(*read*, *isoseg*, *segment_list*, *intron_seg_indx_list*, *exon_seg_indx_list*, *spliced_segment_pair*)
Label the error region for this segment array

Parameters

- **read** (*AlignedSegment*) – TGS read
- **isoseg** (*narray or list*) – Projected segment for this TGS read
- **segment_list** (*list*) – Valid segment list
- **intron_seg_indx_list** (*list*) – List of intron segment index
- **exon_seg_indx_list** (*list*) – List of exon segment index
- **spliced_segment_pair** (*list*) – List of spliced segment pair

Returns Isoform segment array

Return type narray

classmethod filter_iso(*iso*, *segment_list*, *intron_seg_indx_list*, *exon_seg_indx_list*)
Find if the isoform is corrected

Parameters

- **iso** (*Isoform*) – Isoform to task
- **segment_list** (*list*) – Valid segment list
- **intron_seg_indx_list** (*list*) – List of intron segment index
- **exon_seg_indx_list** (*list*) – List of exon segment index

Returns Is corrected or not

Return type bool

classmethod indx2ival(*indx_array*)
Convert index array to index range

classmethod is_incl_segment(*read*, *segment*, *boundary='both'*)
Check if incl-segment boundaries are included

classmethod is_internal_exon(*exon_indx*, *exon_seg_indx_list*)
Judge if TGS internal exon is valid

Parameters

- **exon_indx** (*tuple*) – (start_indx, end_indx)
- **exon_seg_indx_list** (*list*) – Segment internal exon index for all valid internal exon

Returns Is internal exon or not

Return type bool

classmethod is_intron(*intron_indx*, *intron_seg_indx_list*)
Judge if TGS intron is in valid intron list

Parameters

- **intron_indx** (*tuple*) – (start_indx, end_indx)
- **intron_seg_indx_list** (*list*) – Segment intron index for all valid introns

Returns Is intron or not

Return type bool

classmethod is_skip_segment(*read*, *segment*)
Check if skip-segment boundaries are not included

classmethod is_spliced_seg_in_internal_exon(*exon_indx*, *spliced_segment_indx*)
Judge if an internal exon contains spliced segment

class igia.transcript.**TransDiscover**
General method in identify transcript.

classmethod build_compatible_matrix(*isolist*)
Build isoform compatibility matrix

classmethod build_element_seg_indx(*element_list*, *segment_list*)
Find intron index in segment list.

classmethod build_overlap_matrix(*isolist*)
Build isoform overlap matrix

classmethod build_subpath_matrix(*compatible_matrix*, *isolist*)
Build isoform subpath matrix, [i, j] = T => i contains j.

classmethod build_tes_seg_indx(*segment_list*)
Find TES segment index.

classmethod build_tss_seg_indx(*segment_list*)
Find TSS segment index.

classmethod cluster_iso(*iso_list*)
Clustering isoforms by overlap

classmethod complete_iso_by_fl_iso(*isoform*, *fl_segmat*, *tss_indxs*, *tes_indxs*)
Complete isoform with information in full isoform

classmethod complete_partial_isoform(*isoform*, *ta*)
Complete a partial isoform

classmethod complete_partial_isoform_error(*segli*, *minor_list*, *ta*, *s_indx*, *e_indx*)
Try to corrected the errors in a isoform

classmethod complete_partial_isoform_left(*segli*, *minor_list*, *ta*)
Complete the left site missing region of a partial isoform

classmethod complete_partial_isoform_right(*segli*, *minor_list*, *ta*)
Complete the missing region of a partial isoform

classmethod compute_as_cnt(*intron*, *seglink_exon_cnt*, *seglink_intron_cnt*)
Compute read count for an intron

classmethod compute_seg_link_cnt(*ngs_file*, *segli*)
Count reads number supported the given segment pair

classmethod create_intron_path(*intron_nodes*, *start_seg_indx*, *end_seg_indx*)
Create intron path by intron index information

classmethod determin_intron_type(*intron*, *seglink_exon_cnt*, *seglink_intron_cnt*, *segment_list*)
Find if a intron could be an IR

classmethod element2segment(*intron_list*, *exon_list*, *trans_ival*)
Build transcript segments by transcript elements information.

classmethod enum_intron_path(*intron_seg_indx_list*, *start_seg_indx*, *end_seg_indx*, *segment_list*, *seglink_cnt*)
Enumerate intron graph in certain region.

classmethod filter_compatible_with_overlap(*indxs*, *clusters*, *overlap_matrix*)
Only overlapped and compatible isoforms can be merged

classmethod invalid_iso_is_subpath(*iso*, *fl_isos*)
If a invalid isoform is a subpath for a full length isoform

classmethod is_compatible(*iso1*, *iso2*)
Judge if two isoform are compatibility

classmethod is_overlap(*iso1*, *iso2*)
Judge if two isoform are overlapped

classmethod is_subpath(*iso_indx*, *subpath_matrix*)
Judge whether a isoform is a subpath of another isoform or not.

classmethod merge_cluster_isoforms(*indxs*, *isolist*, *tss_indxs*, *tes_indxs*)
Merge multiple isoforms from same cluster

Parameters

- **indxs** (*list*) – Isoform indx to be merged
- **isolist** (*list*) – Isoform list
- **tss_indxs** (*list*) – List of TSS segment index
- **tes_indxs** (*list*) – List of TES segment index

Returns IsoM

Return type *Isoform*

classmethod merge_nfl_isoforms(*iso_indxs*, *compatible_matrix*, *overlap_matrix*, *isolist*, *tss_indxs*, *tes_indxs*)
Clustering non-full-length isoforms into clusters and then merge isoforms in clusters

classmethod rescue_isoform(*invalid_iso*, *isoF*, *intron_seg_indx_list=None*, *exon_seg_indx_list=None*, *segment_list=None*)
Rescue invalid full-length TGS isoform by fixing errors from most similar isoform

classmethod rescue_junction(*invalid_iso*, *intron_seg_indx_list*, *exon_seg_indx_list*, *segment_list*)
Rescue junction by validated information

classmethod search_nfl_cluster(*iso_indxs*, *compatible_matrix*)
The extremely complete subgraph of compatibility matrix can be seem as a cluster of non-full-length isoforms

Parameters

- **iso_indxs** (*narray*) – Non-full length isoform index array
- **compatible_matrix** (*narray*) – Isoform compatibility matrix

Returns Set of isoform index in one cluster

Return type set

classmethod similar_score(*segary1*, *segary2*)
Compute similar score between two segment array

classmethod split_iso_by_subpath(*test_iso_list*, *parent_iso_list*)
Split isoform list into subpath and not subpath.

Parameters

- **test_iso_list** (*list*) – Isoform list to split
- **parent_iso_list** (*list*) – Isoform list to reference

Returns (subpath, not_subpath)

Return type tuple

class igia.transcript.**TgsTransDiscover**
Identify transcript with TGS data

classmethod compute_iso_overlap_fraction(*read*, *segment*)
Compute the overlap fraction between a read and a segment

classmethod refine(*tgs_read_list*, *segment_list*, *intron_seg_indx_list*, *exon_seg_indx_list*, *spliced_segment_pair*, *tgs_overlap_fraction_threshold*, *trans_ival*)
Project TGS read to segment list, refine and create Isoform

igia.transcript.**identify_transcript**(*gene*, *ann=None*)
Method to identity transcript