CrossMap Documentation

Release 0.1.1

Liguo Wang

October 23, 2013

- CrossMap is a program for convenient conversion of genome coordinate (or annotation files) between *different assemblies* (such as Human hg18 (NCBI36) <-> hg19 (GRCh37), Mouse mm9 (MGSCv37) <-> mm10 (GRCm38)).
- As been said, CrossMap is designed to liftover genome coordinates between assemblies. It's *not* a program for aligning sequence to reference genome.
- We *do not* recommend using CrossMap to convert genome coordinates between species.

CHAPTER

ONE

WHY CROSSMAP ?

Full genome sequencing, especially mammalian genomes like human, is a long period of work requiring continuous efforts, and therefore reference genome assemblies are subject to change/refine from time to time. Generally, researchers need to convert results analyzed according to old assembly to newer version or vice versa to facilitate meta-analyses, direct comparison as well as data integration and visualization. Several useful conversion tools have been developed:

- UCSC liftover tool only supports BED input.
- Galaxy (Based on UCSC liftover tool) supports BED, GFF, GTF input.
- Ensembl assembly converter supports BED, GFF, GTF, PSL input, but output is GFF only.
- pyliftover "only does conversion of point coordinates, that is, unlike liftOver, it does not convert ranges, nor does it provide any special facilities to work with BED files".

but all of them can only convert genome interval files in BED or GFF format and none of them can convert files in BAM/SAM or BigWig format, which are the most widely used file format to represent high-throughput sequencing data including RNA-seq, ChIP-seq, DNA-seq, etc.

CHAPTER TWO

DOWNLOAD

- Download CrossMap program from here
- Download test data from here

CHAPTER THREE

INSTALLATION

Prerequisite:

1. gcc
2. python2.7 or newer
3. numpy
4. cython

```
$ tar zxf CrossMap-VERSION.tar.gz

$ cd CrossMap-VERSION

# install CrossMap to default location. In Linux/Unix, this location is some thing like: /home/user/l
$ python setup.py install

# or you can install CrossMap to a specified location:
$ python setup.py install --root=/home/user/CrossMap

# setup PYTHONPATH. Skip this step if CrossMap was installed to default location.
$ export PYTHONPATH=/home/user/CrossMap/usr/local/lib/python2.7/site-packages:$PYTHONPATH.

# Skip this step if CrossMap was installed to default location.
$ export PATH=/home/user/CrossMap/usr/local/bin:$PATH
```

NOTE:

1. Due to intensive computation, CrossMap is designed to run on Linux/Unix and Mac OS. Some modules may not work properly on Windows.

CHAPTER

FOUR

INPUT AND OUTPUT

CrossMap needs 2 input files. chain format file describing genom-wide pairwise alignment between assemblies and the file containing genome coordinates that you want to convert to different assembly.

4.1 Chain file

Example of chain file:

```
chain 4900 chrY 58368225 + 25985403 25985638 chr5 151006098 - 43257292 43257528 1
 9       1       0
 10      0       5
 61      4       0
 16      0       4
 42      3       0
 16      0       8
 14      1       0
 3       7       0
 48

 chain 4900 chrY 58368225 + 25985406 25985566 chr5 151006098 - 43549808 43549970 2
 16      0       2
 60      4       0
 10      0       4
 70
```

UCSC prebuilt most commonly used chain files:

- Human (*Homo sapiens*)
- hg19ToHg18.over.chain.gz (Chain file needed to convert hg19 to hg18)
- hg19ToHg17.over.chain.gz (Chain file needed to convert hg19 to hg17)
- hg18ToHg19.over.chain.gz (Chain file needed to convert hg18 to hg19)
- hg18ToHg17.over.chain.gz (Chain file needed to convert hg18 to hg17)
- hg17ToHg19.over.chain.gz (Chain file needed to convert hg17 to hg19)
- hg17ToHg18.over.chain.gz (Chain file needed to convert hg17 to hg18)
- Mouse (*Mus musculus*)
- mm10ToMm9.over.chain.gz (Chain file needed to convert mm10 to mm9)
- mm9ToMm10.over.chain.gz (Chain file needed to convert mm9 to mm10)
- mm9ToMm8.over.chain.gz (Chain file needed to convert mm9 to mm8)

Chain file of other species can be downloaded from here

4.2 User Input file

1. BAM or SAM format.
2. BED or BED-like format. BED file must has at least 3 columns (‘chrom’, ‘start’, ‘end’).
3. Wiggle. Both “variableStep” and “fixedStep” wiggle line are supported.
4. BigWig format.
5. GFF or GTF format.
6. VCF format.

4.3 Output

Format of Output files depends on the input format

Input_format	Output_format
BED	BED (Genome coordinates will be updated to the target assembly)
BAM	BAM (Genome coordinates, all SAM flags, insert size will be updated accordingly)
SAM	SAM (Genome coordinates, all SAM flags, insert size will be updated accordingly)
Wiggle	bedGraph (if wigToBigWig executable does not exist)
Wiggle	BigWig (if wigToBigWig executable exists)
BigWig	bedGraph (if wigToBigWig executable does not exist)
BigWig	BigWig (if wigToBigWig executable exists)
GFF	GFF (Genome coordinates will be updated to the target assembly)
GTF	GTF (Genome coordinates will be updated to the target assembly)
VCF	VCF (Genome coordinates and reference allele will be updated to the target assembly)

CHAPTER

FIVE

USAGE

Run CrossMap.py without any arguments will print help message:

```
# run CrossMap without argument
$ python CrossMap.py
```

Screen output:

```
Program: CrossMap (v0.1.1)

Description:
  CrossMap is a program for convenient conversion of genome coordinates
  and genomeannotation files between assemblies (eg. lift from human
  hg18 to hg19 or vice versa).It support file in BAM, SAM, BED, Wiggle,
  BigWig, GFF, GTF, VCF, etc.

Usage: CrossMap.py <command> [options]

  bam  convert alignment file in BAM or SAM format.
  bed  convert genome cooridnate or annotation file in BED or BED-like format.
  bigwig       convert genome coordinate file in BigWig format.
  gff  convert genome cooridnate or annotation file in GFF or GTF format.
  vcf  convert genome coordinate file in VCF format.
  wig  convert genome coordinate file in Wiggle, or bedGraph format.
```

Run CrossMap.py with command keyword will print help message for that command:

```
$ python CrossMap.py bed
```

Screen output:

```
Usage:
  CrossMap.py bed input_chain_file input_bed_file [output_file]

Description:
  "input_chain_file" and "input_bed_file" can be regular or compressed
  (*.gz, *.Z, *.z, *.bz, *.bz2, *.bzip2) file, local file or URL
  (http://, https://, ftp://) pointing to remote file. BED file must
  have at least 3 columns (chrom, start, end) and no more than 12
  columns. If  no "output_file" was specified, output will be directed
  to screen (console). BED format:
  http://genome.ucsc.edu/FAQ/FAQformat.html#format1

Example:
  CrossMapy.py bed hg18ToHg19.over.chain.gz test.hg18.bed test.hg19.bed
  # write output to "test.hg19.bed"
```

```
Example:
  CrossMapy.py bed hg18ToHg19.over.chain.gz test.hg18.bed
  # write output to screen
```

5.1 Convert BED format files

Standard BED format has 12 columns, but CrossMap also support BED-like formats:

- BED3: The first 3 columns ("chrom", "start", "end") of BED format file.
- BED6: The first 6 columns ("chrom", "start", "end", "name", "score", "strand")of BED format file.
- Other: Format has at least 3 columns ("chrom", "start", "end") and no more than 12 columns. All other columns are arbitrary.

NOTE:

1. For BED-like formats mentioned above, CrossMap only updates "chrom (1st column)", "start (2nd column) ", "end (3rd column) " and "strand" (if any). All other columns will keep AS-IS.
2. Lines starting with '#', 'browser', 'track' will be skipped.
3. Lines will less than 3 columns will be skipped.
4. 2nd-column and 3-column must be integer, otherwise skipped.
5. "+" strand is assumed if no strand information was found.
6. For standard BED format (12 columns). If any of the defined exon blocks cannot be uniquely mapped to target assembly, the whole entry will be skipped.
7. "input_chain_file" and "input_bed_file" can be regular or compressed (.gz, .Z, .z, .bz, .bz2, .bzip2) file, local file or URL (http://, https://, ftp://) pointing to remote file.
8. If output_file was not specified, results will be printed to screen (console). In this case, the original bed entries were also printed out.
9. If input region cannot be consecutively mapped target assembly, it will be split.

Example (run CrossMap with **no** output_file specified):

```
$ python CrossMap.py bed hg18ToHg19.over.chain.gz test.hg18.bed3
```

Conversion results printed to screen directly (column1-3 are hg18 based, column5-7 are hg19 based):

```
chr1   142614848       142617697       ->      chr1    143903503       143906352
chr1   142617697       142623312       ->      chr1    143906355       143911970
chr1   142623313       142623350       ->      chr1    143911971       143912008
chr1   142623351       142626523       ->      chr1    143912009       143915181
chr1   142633862       142633883       ->      chr1    143922520       143922541
chr1   142633884       142636152       ->      chr1    143922542       143924810
chr1   142636152       142636326       ->      chr1    143924813       143924987
chr1   142636339       142636391       ->      chr1    143925000       143925052
chr1   142636392       142637362       ->      chr1    143925052       143926022
chr1   142637373       142639738       ->      chr1    143926033       143928398
chr1   142639739       142639760       ->      chr1    143928399       143928420
chr1   142639761       142640145       ->      chr1    143928421       143928805
chr1   142640153       142641149       ->      chr1    143928813       143929809
```

Example (run CrossMap with output_file **(test.hg19.bed3)** specified):

```
$ python CrossMap.py bed hg18ToHg19.over.chain.gz test.hg18.bed3 test.hg19.bed3

$ cat test.hg19.bed3
chr1   143903503       143906352
chr1   143906355       143911970
chr1   143911971       143912008
chr1   143912009       143915181
chr1   143922520       143922541
chr1   143922542       143924810
chr1   143924813       143924987
chr1   143925000       143925052
chr1   143925052       143926022
chr1   143926033       143928398
chr1   143928399       143928420
chr1   143928421       143928805
chr1   143928813       143929809
```

Example (one input region was split because it cannot be consecutively mapped target assembly):

```
$ python CrossMap.py bed hg18ToHg19.over.chain.gz test.hg18.bed3

chr10  81346644        81349952        +       ->      chr10   81356692        81360000        +
chr10  81349952        81364937        +       ->      chr10   81360000        81374985        +
chr10  81364952        81365854        +       ->      chr10   81375000        81375902        +
chr10  81365875        81369946        +       ->      chr10   81375929        81380000        +
chr10  81369946        81370453        +       ->      chr10   81380000        81380507        +
chr10  81370483        81371363        +       ->      chr10   81380539        81381419        +
chr10  81371363        81371365        +       ->      chr10   62961832        62961834        +
chr10  81371412        81371432        +       (split.1:chr10:81371412:81371422:+)     chr10   62961
chr10  81371412        81371432        +       (split.2:chr10:81371422:81371432:+)     chrX    63278
```

5.2 Convert BAM/SAM format files

Both SAM and BAM are supported. Input file must have header section. If input is BAM file, it should be sorted and indexed using samTools

Typing command without any arguments will print help message:

```
$ python CrossMap.py bam
```

Screen output:

```
Usage: CrossMap.py bam input_chain_file input_bam_file output_file

Options:
  -m INSERT_SIZE        Average insert size of pair-end sequencing (bp).
                        [default=200.0]
  -s INSERT_SIZE_STDEV  Stanadard deviation of insert size. [default=30.0]
  -t INSERT_SIZE_FOLD   A mapped pair is considered as "proper pair" if both
                        ends mapped to different strand and the distance
                        between them is less then '-t' * stdev from the mean.
                        [default=3.0]
```

NOTE:

1. Input is BAM or SAM format file. Output format depends on input format. (i.e BAM -> BAM, SAM -> SAM)
2. Alignments that are failed to convert will be saved in “.*unmap.bam” or* “.unmap.bam”

3. Header section will be updated to target assembly.
4. Genome coordinates and all SAM flags in alignment section will be updated to target assembly.
5. Optional fields in alignment section will not update.

5.3 Convert Wiggle format files

Input file is Wiggle format. If ‘wigToBigWig‘ executable is found, output is BigWig format, otherwise output will be in bedGraph format. bedGraph file can be converted into BigWig using wigToBigWig tool.

Typing command without any arguments will print help message:

```
$ python2.7 CrossMap.py  wig
```

Screen output:

```
Usage:
  CrossMap.py wig input_chain_file input_wig_file output_prefix

Description:
  "input_chain_file" can be regular or compressed (*.gz, *.Z, *.z, *.bz, *.bz2,
  *.bzip2) file, local file or URL (http://, https://, ftp://) pointing to remote
  file.  Both "variableStep" and "fixedStep" wiggle lines are supported. Wiggle
  format: http://genome.ucsc.edu/goldenPath/help/wiggle.html

Example:
  CrossMapy.py wig hg18ToHg19.over.chain.gz test.hg18.wig test.hg19
```

NOTE:

1. To improve performance, this script calls GNU “sort” command internally. If “sort” command does not exist, CrossMap will exit.

5.4 Convert BigWig format files

Input file is BigWig format. If ‘wigToBigWig‘ executable is found, output is BigWig format, otherwise output will be in bedGraph format. bedGraph file can be converted into BigWig using wigToBigWig tool.

Typing command without any arguments will print help message:

```
$ python2.7 CrossMap.py  bigwig
```

Screen output:

```
Usage:
  CrossMap.py bigwig input_chain_file input__bigwig_file output_prefix

Description:
  "input_chain_file" can be regular or compressed (*.gz, *.Z, *.z, *.bz, *.bz2,
  *.bzip2) file, local file or URL (http://, https://, ftp://) pointing to remote
  file. Bigwig format: http://genome.ucsc.edu/goldenPath/help/bigWig.html

Example:
  CrossMapy.py bigwig hg18ToHg19.over.chain.gz test.hg18.bw test.hg19
```

NOTE:

1. To improve performance, this script calls GNU "sort" command internally. If "sort" command does not exist, CrossMap will exit.

5.5 Convert GFF/GTF format files

Typing command without any arguments will print help message:

```
$ python2.7 CrossMap.py  gff
```

Screen output:

```
Usage:
  CrossMap.py gff input_chain_file input_gff_file output_file

Description:
  "input_chain_file" can be regular or compressed (*.gz, *.Z, *.z, *.bz, *.bz2,
  *.bzip2) file, local file or URL (http://, https://, ftp://) pointing to remote
  file. input file must be in GFF or GTF format. GFF format:
  http://genome.ucsc.edu/FAQ/FAQformat.html#format3 GTF format:
  http://genome.ucsc.edu/FAQ/FAQformat.html#format4

Example:
  CrossMapy.py gff hg18ToHg19.over.chain.gz test.hg18.gff test.hg19.gff  # write
  output to "test.hg19.bed"

Example:
  CrossMapy.py gff hg18ToHg19.over.chain.gz test.hg18.gff  # write output to
  screen
```

NOTE:

1. Each feature (exon, intron, UTR, etc) is processed separately and independently, and we do NOT check if features originally belonging to the same gene were converted into the same gene.
2. If user want to liftover gene annotation files, use BED12 format.
3. If no output file was specified, output will be printed to screen (console).

5.6 Convert VCF format files

Typing command without any arguments will print help message:

```
$ python2.7 CrossMap.py  gff
```

Screen output:

```
usage:
  CrossMap.py vcf input_chain_file input_VCF_file ref_genome_file output_file

Description:
  "input_chain_file" and "input_VCF_file" can be regular or compressed (*.gz, *.Z,
  *.z, *.bz, *.bz2, *.bzip2) file, local file or URL (http://, https://, ftp://)
  pointing to remote file. "ref_genome_file" is genome sequence file of 'target
  assembly' in FASTA foramt.

Example:
  CrossMapy.py vcf hg18ToHg19.over.chain.gz test.hg18.vcf hg19.fa hg19.out
```

NOTE:

1. Genome coordinates and reference allele will be updated to target assembly.

CHAPTER

SIX

LICENSE

CrossMap is distributed under GNU General Public License

CHAPTER SEVEN

CONTACT

- Wang.Liguo AT mayo.edu