GenomicTools v2.0: a computational platform for developing high-throughput analytics in genomics

Aristotelis Tsirigos, Niina Haiminen, Erhan Bilal, Filippo Utro
IBM Thomas J. Watson Research Center, PO Box 218,
Yorktown Heights, NY 10598, U.S.A

Contact: atsirigo@us.ibm.com

Version 2.0.0
Copyright (c) 2011 IBM Corporation. All rights reserved.
# Table of Contents

1  Overview .......................................................................................................................... 1

1.1  Introduction .................................................................................................................... 1

1.2  Features .......................................................................................................................... 1

1.3  Definitions ....................................................................................................................... 2

1.4  Supported file formats .................................................................................................. 3

1.5  Summary of tools .......................................................................................................... 3

2  C++ API for developers .................................................................................................... 6

2.1  The GenomicInterval and GenomicRegion classes ...................................................... 6

2.2  The GenomicRegionSet class ....................................................................................... 7

2.3  The GenomicRegionSetScanner class ........................................................................ 9

2.4  The GenomicRegionSetOverlaps class and its extensions ....................................... 10

3  Installation ....................................................................................................................... 12

4  The GenomicTools suite ................................................................................................... 13

4.1  The genomic_regions tool .......................................................................................... 13

4.1.1  genomic_regions align .......................................................................................... 14

4.1.2  genomic_regions bed ............................................................................................. 15

4.1.3  genomic_regions bounds ....................................................................................... 16

4.1.4  genomic_regions center ......................................................................................... 17

4.1.5  genomic_regions connect ...................................................................................... 18

4.1.6  genomic_regions diff ............................................................................................. 19

4.1.7  genomic_regions dist ............................................................................................. 20

4.1.8  genomic_regions divide .......................................................................................... 21

4.1.9  genomic_regions fix ............................................................................................... 22
4.1.10  genomic_regions gdist .............................................................................................. 23
4.1.11  genomic_regions int ..................................................................................................... 24
4.1.12  genomic_regions inv ..................................................................................................... 25
4.1.13  genomic_regions link .................................................................................................... 26
4.1.14  genomic_regions n .......................................................................................................... 27
4.1.15  genomic_regions pos ...................................................................................................... 28
4.1.16  genomic_regions reg ..................................................................................................... 29
4.1.17  genomic_regions rnd ..................................................................................................... 30
4.1.18  genomic_regions select ................................................................................................. 31
4.1.19  genomic_regions shift .................................................................................................... 32
4.1.20  genomic_regions shiftp ................................................................................................. 33
4.1.21  genomic_regions shuffle ............................................................................................... 34
4.1.22  genomic_regions sort ..................................................................................................... 35
4.1.23  genomic_regions split .................................................................................................... 36
4.1.24  genomic_regions strand ............................................................................................... 37
4.1.25  genomic_regions test ..................................................................................................... 38
4.1.26  genomic_regions union .................................................................................................. 39
4.1.27  genomic_regions wig ..................................................................................................... 40
4.1.28  genomic_regions win ..................................................................................................... 41
4.1.29  genomic_regions x ......................................................................................................... 42
4.2   The genomic_overlaps tool ................................................................................................. 43
4.2.1   genomic_overlaps count ............................................................................................... 44
4.2.2   genomic_overlaps coverage ........................................................................................... 45
4.2.3   genomic_overlaps density ............................................................................................. 46
4.2.4   genomic_overlaps intersect .......................................................................................... 47
4.2.5  genomic_overlaps offset .................................................................................. 48
4.2.6  genomic_overlaps overlap................................................................................. 49
4.2.7  genomic_overlaps subset ................................................................................... 50
4.3  The genomic_scans tool ....................................................................................... 51
  4.3.1  genomic_scans counts ...................................................................................... 51
  4.3.2  genomic_scans peaks ....................................................................................... 52
4.4  Permutation tests .................................................................................................... 53
4.5  Vector operations .................................................................................................... 55
4.6  Matrix operations .................................................................................................... 56
5  Case study: a simple ChIP-seq pipeline ................................................................. 57
  5.1  Creating ChIP-seq read profiles .......................................................................... 57
  5.2  Creating ChIP-seq read density heatmaps ............................................................ 59
  5.3  Creating window-based read densities ................................................................. 60
  5.4  Identifying window-based peaks ......................................................................... 61
  5.5  Identifying enriched Gene Ontology terms ......................................................... 62
6  References .................................................................................................................... 65
1 Overview

1.1 Introduction

GenomicTools is flexible computational platform for the analysis and manipulation of high-throughput sequencing data such as DNA-seq, RNA-seq, ChIP-seq and MethylC-seq. GenomicTools implements a variety of mathematical operations between sets of genomic regions thereby enabling the prototyping of computational pipelines that can address a wide spectrum of tasks from preprocessing and quality control to meta-analyses. For example, the user can easily create average read profiles across transcriptional start sites or enhancer sites, quickly prototype customized peak discovery methods for ChIP-seq experiments, perform genome-wide statistical tests such as enrichment analyses, design controls via appropriate randomization schemes, among other applications. In addition to enabling rapid prototyping, the GenomicTools platform is designed to analyze large-datasets of any size by minimizing memory requirements. GenomicTools supports the widely used BED, GFF and SAM formats to facilitate visualization as well as integration with existing platforms and pipelines such as the UCSC Genome Browser [1], Galaxy [2], Bioconductor [3] and BEDTools [4]. The inspiration for this work can be traced to our previous studies on repeat elements [5][6][7] where the GenomicTools suite was first used.

1.2 Features

The GenomicTools platform, while similar in motivation to other tools, such as BEDTools, it is in several aspects more general and it addresses several open issues. We summarize the novelty of GenomicTools below:

- **Novel operations:** in GenomicTools the focus is not simply on overlap computations, but is instead designed to perform: (a) a variety of simple mathematical operations on sets genomic intervals (as a preprocessing step), and (b) a variety of complex operations, such as overlap, offset or scanning computations.

- **Relaxed dataset restrictions:** GenomicTools allows several of its operations to operate on sets of genomic regions rather than sets of single genomic intervals, updating all the major fields for all supported formats; for example it makes full use of all “exons” in BED entries and it updates the thickStart and thickEnd variables.

- **Full stream-computing design:** in GenomicTools files are typically processed as streams: this minimizes memory requirements and allows the simultaneous processing of several files (e.g. different replicates, patient samples, etc.).
• **C++ API:** *GenomicTools* command-line operations are implemented as C++ class methods in a convenient API, which can be used by developers to write new applications entirely in C++, e.g. novel peak finders.

• **Auxiliary tools:** *GenomicTools* offers a set of auxiliary command-line tools (permutation_test, vectors, and matrix) to facilitate the construction of command-line pipelines as they implement basic mathematical and statistical operations on vectors and matrices.

• **Performance:** *GenomicTools* improves performance when compared to similar tools, both in terms of time and memory requirements.

### 1.3 Definitions

A *genomic interval* is a tuple: `<chromosome, strand, start position, end position>`.

A *genomic region* is an ordered set of genomic intervals. Note that this is a rather broad definition, which allows for the inclusion of genomic intervals from different chromosomes and/or strands, as well as intervals that overlap. In GenomicTools, this definition of genomic regions is implemented in the REG file format, which we introduce in the next section.

Genomic regions are characterized by several properties. A genomic region is *compatible* if and only if all its intervals are in the same chromosome and (optionally) strand. A genomic region is *sorted* if and only if its intervals are sorted first by chromosome, then optionally by strand and finally by start position. A genomic region is *non-overlapping* if and only if its intervals are non-overlapping in all pairwise combinations. A genomic region is a *single-interval region* if and only if it contains exactly one interval. Operations on genomic regions may require that certain properties be satisfied before they can be successfully executed.

A *genomic region set* is an ordered set of genomic regions. A genomic region set is *sorted* if and only if its regions are single-interval regions and they appear in the sort order described above. As before, operations on genomic region sets may require that certain properties be satisfied before they can be successfully executed.

In GenomicTools, all input files contain a single genomic region set as defined above. Every line of these files corresponds to a single genomic region possibly annotated with additional information, such as labels, depending on the particular file format (see next section).
1.4 Supported file formats

The GenomicTools platform uses the REG file format (described below) as the preferred format for input files, although it also supports the BED\(^1\), GFF\(^2\) and SAM\(^3\) file formats. Input files can also be converted into WIG format\(^4\). The REG format is an attempt to distill the minimum common information from the BED/GFF/SAM formats while allowing for the more general definition of a genomic region as defined above. Each line in a REG file represents a labeled genomic region, where the label is separated from the genomic region via a \(<\text{TAB}>\) character. A simple REG file representing a set of RNAseq reads is shown below:

<table>
<thead>
<tr>
<th>Read#1&lt;TAB&gt;1 + 100 149</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read#2&lt;TAB&gt;1 + 102 151</td>
</tr>
<tr>
<td>...</td>
</tr>
<tr>
<td>Read#N&lt;TAB&gt;Y – 10001 10050</td>
</tr>
</tbody>
</table>

Another example is the following REG file carrying information on gene exons (note that every line is a set of genomic intervals):

<table>
<thead>
<tr>
<th>Gene#1&lt;TAB&gt;1 + 160446 161690 1 + 161314 161525</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
</tr>
<tr>
<td>Gene#N&lt;TAB&gt;Y - 279704 279708 Y - 279741 279839 Y - 279911 279916</td>
</tr>
</tbody>
</table>

Note that this format is a generalization of the BED format because it allows overlapping intervals within a given region (Gene#1 in the above example). This is particularly useful when we need to group exons of a set of transcript isoforms of the same gene. Also, it allows for intervals from different chromosomes and strands to be grouped in each line, and this helps represent gene fusions and interchromosomal associations.

In terms of C++ implementation, each genomic region (i.e. each line in an input file) is stored as an instance of the GenomicRegion class or its derived classes for BED, GFF and SAM formats. The entire file is stored as an instance of the GenomicRegionSet class, although not necessarily fully loaded in memory.

1.5 Summary of tools

The genomic_regions tool is designed to allow the user to perform basic operations for manipulating genomic region files. These are: (a) line-based operations, such as shifting, shuffling, sorting, and modifying genomic regions, and (b) file-based operations such as linking, or inverting. A summary of

1. http://genome.ucsc.edu/FAQ/FAQformat.html#format1
the available operations of the genomic_regions tool is shown in Table 1. Importantly, all major fields of the supported formats are appropriately updated.

The genomic_overlaps tool allows the user to compute various measures of overlaps between sets of regions. This is achieved by providing a set of operations and a variety of options for: (a) finding regions that match or partially overlap, (b) counting the number of matches, (c) calculating densities of matched regions, and (d) computing relative distances (i.e. offsets) between overlapping regions. Applications include computation of gene expression, construction of average read profiles across gene bodies, transcriptional start sites (TSSs) etc, and enrichment analyses of virtually any genomic dataset, such as genes of specific functional categories, repeat types, SNPs, cancer-associated regions. The available operations of the genomic_overlaps tool are summarized in Table 2.

The genomic_scans tool (scan_reads in version 1.3) is used for window-based computations such as peak discovery (see Table 3). The command-line version offers several parameters for controlling the window size, statistical tests etc. Users with basic C/C++ skills can easily modify the source code to perform the statistical test of their choice using the GenomicRegionSetScanner class described in section 2.3 (page 1).

Table 1: Summary of operations of the genomic_regions command-line tool.

<table>
<thead>
<tr>
<th>Operation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>align</td>
<td>Aligns sequences to reference genome (line-based)</td>
</tr>
<tr>
<td>bed</td>
<td>Converts input regions to BED format (line-based)</td>
</tr>
<tr>
<td>bounds</td>
<td>Checks interval against chromosome bounds and removes invalid intervals (line-based)</td>
</tr>
<tr>
<td>center</td>
<td>Prints center interval (line-based)</td>
</tr>
<tr>
<td>connect</td>
<td>Connects intervals from minimum start to maximum stop (line-based)</td>
</tr>
<tr>
<td>diff</td>
<td>Computes the difference between successive intervals (line-based)</td>
</tr>
<tr>
<td>dist</td>
<td>Computes distances between successive intervals (line-based)</td>
</tr>
<tr>
<td>divide</td>
<td>Divides intervals in the middle (line-based)</td>
</tr>
<tr>
<td>fix</td>
<td>Removes invalid intervals, i.e. start&lt;1 or start&gt;stop (line-based)</td>
</tr>
<tr>
<td>gdist</td>
<td>Computes distances of successive regions (file-based)</td>
</tr>
<tr>
<td>int</td>
<td>Computes the intersection of input intervals (line-based)</td>
</tr>
<tr>
<td>inv</td>
<td>Inverts regions given the genome chromosomal boundaries (file-based)</td>
</tr>
<tr>
<td>link</td>
<td>Links consecutive regions to produce a non-overlapping set (file-based)</td>
</tr>
<tr>
<td>n</td>
<td>Computes total interval length, including possible overlaps (line-based)</td>
</tr>
<tr>
<td>pos</td>
<td>Modifies interval start/stop positions (line-based)</td>
</tr>
<tr>
<td>reg</td>
<td>Converts to REG format (line-based)</td>
</tr>
<tr>
<td>rnd</td>
<td>Randomizes region across entire genome (line-based)</td>
</tr>
<tr>
<td>select</td>
<td>Selects a subset of intervals according to their relative start positions (line-based)</td>
</tr>
<tr>
<td>shift</td>
<td>Shifts interval start/stop positions (line-based)</td>
</tr>
<tr>
<td>shiftp</td>
<td>Shifts interval 5'/3' positions (line-based)</td>
</tr>
<tr>
<td>shuffle</td>
<td>Shuffles intervals within given reference region (line-based)</td>
</tr>
<tr>
<td>sort</td>
<td>Sorts intervals (line-based)</td>
</tr>
<tr>
<td>split</td>
<td>Splits regions into their intervals which are printed on separate lines (line-based)</td>
</tr>
</tbody>
</table>
Table 2: Summary of operations of the genomic_overlaps command-line tool.

<table>
<thead>
<tr>
<th>Operation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>count</td>
<td>Counts the number of overlapping test regions per reference region.</td>
</tr>
<tr>
<td>coverage</td>
<td>Calculates the depth coverage (i.e. the total number of overlapping nucleotides) per reference region.</td>
</tr>
<tr>
<td>density</td>
<td>Computes the density (i.e. the coverage divided by the size of the reference region) of overlaps per reference region.</td>
</tr>
<tr>
<td>intersect</td>
<td>Computes the intersection between all pairs of test and reference regions</td>
</tr>
<tr>
<td>offset</td>
<td>Computes the distances of test regions from their overlapping reference regions</td>
</tr>
<tr>
<td>overlap</td>
<td>Finds the overlaps between all pairs of test and reference regions</td>
</tr>
<tr>
<td>subset</td>
<td>Picks a subset of test regions depending on their overlap with reference regions</td>
</tr>
</tbody>
</table>

Table 3: Summary of operations of the genomic_scans command-line tool.

<table>
<thead>
<tr>
<th>Operation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>counts</td>
<td>Determines input read counts in sliding windows of reference regions.</td>
</tr>
<tr>
<td>peaks</td>
<td>Scans input reads to identify peaks.</td>
</tr>
</tbody>
</table>
2 C++ API for developers

Users with basic C++ skills can make use of the genomic_intervals library, particularly for window-based computations and overlaps. All implemented classes are fully documented using Doxygen http://www.doxygen.org (see Figure 1 for a snapshot). Access to full documentation is provided along with the source code distribution. In the following sections, we describe the main classes that are used to represent the genomic data and perform the various operations.

2.1 The GenomicInterval and GenomicRegion classes

The GenomicInterval class simply implements the notion of a genomic interval, i.e. an interval annotated with chromosome and strand information. The GenomicRegion class implements the notion of a genomic region (in REG format), i.e. a labeled ordered set of genomic intervals, and corresponds to one single line in the input file. The genomic intervals are stored as C++ STL vectors, but there is also an option of C++ STL lists. This class has a series of constructors, which create genomic regions from an input file (accessed via the FileBuffer class), or from a character array.

The methods of this class are classified into four categories:
(a) read & print methods: read and print genomic intervals in various formats
(b) get & set methods: retrieve and set class variables, such as label, chromosome, etc.
(c) check & compare methods: obtain information about region properties (sorted, compatible, etc.), and their relationship with other regions (overlaps, order, etc)
(d) operations: execute operations between or within regions, such as union, difference, etc.

This GenomicRegion class contains just enough information for the minimal requirements of the REG format. Most methods described above are implemented as virtual so as to allow for class extensions which provide full support for BED (GenomicRegionBED class), GFF (GenomicRegionGFF class) and SAM (GenomicRegionSAM class) formats. The virtual methods are redefined – when necessary – for each derived class in order to properly read, print and update the extra variables of each format. Additionally, for each format, we provide simple classes whose goal is to only read in the corresponding format and immediately convert it into the REG format. These classes are used when the extra variables of the input format are not needed for a particular computation, and can save both time and memory resources. These classes are: GenomicRegionBEDToREG, GenomicRegionGFFToREG and GenomicRegionSAMToREG.

2.2 The GenomicRegionSet class
The GenomicRegionSet class implements the notion of a set of genomic regions which corresponds to an entire input file, for example the set of known genes, aligned reads from a sequencing experiment, etc. The main data element stored in this class is an array of instances of the GenomicRegion class (or any of its derived classes). The input file can be read from the standard input to facilitate pipelined execution, and is not necessarily loaded fully in memory in order to minimize memory requirements: essentially, the input files are read and processed sequentially one line at a time, and the data for each line is discarded when no longer needed for the particular computation. In the next section we show an example of allocating an instance of this class. The operations of the genomic_regions command-line tool summarized in Table 1 are implemented as methods in this class. Developers can use Table 4 as a reference for the methods that correspond to each operation of the genomic_regions tool.
## Table 4: Class methods corresponding to each operation of the genomic_regions command-line tool.

<table>
<thead>
<tr>
<th></th>
<th>class GenomicRegionSet</th>
<th>class GenomicRegion</th>
<th>class GenomicInterval</th>
</tr>
</thead>
<tbody>
<tr>
<td>genomic_regions align</td>
<td>RunAlign()</td>
<td>RunAlign()</td>
<td>---</td>
</tr>
<tr>
<td>genomic_regions bed</td>
<td>RunConvertToBED()</td>
<td>PrintBEDFormat()</td>
<td>---</td>
</tr>
<tr>
<td>genomic_regions bounds</td>
<td>RunBounds()</td>
<td>ApplyBounds()</td>
<td>ApplyBounds()</td>
</tr>
<tr>
<td>genomic_regions center</td>
<td>RunCenter()</td>
<td>Center()</td>
<td>not implemented yet</td>
</tr>
<tr>
<td>genomic_regions connect</td>
<td>RunConnect()</td>
<td>Connect()</td>
<td>---</td>
</tr>
<tr>
<td>genomic_regions diff</td>
<td>RunDiff()</td>
<td>Diff()</td>
<td>---</td>
</tr>
<tr>
<td>genomic_regions dist</td>
<td>RunCalcDistances()</td>
<td>RunCalcDistances()</td>
<td>CalcDistanceFrom()</td>
</tr>
<tr>
<td>genomic_regions divide</td>
<td>RunDivide()</td>
<td>Divide(), RunDivide()</td>
<td>---</td>
</tr>
<tr>
<td>genomic_regions fix</td>
<td>RunFix()</td>
<td>Fix()</td>
<td>CheckValid()</td>
</tr>
<tr>
<td>genomic_regions int</td>
<td>RunIntersection()</td>
<td>Intersect()</td>
<td>---</td>
</tr>
<tr>
<td>genomic_regions n</td>
<td>RunSize()</td>
<td>RunSize(), GetSize()</td>
<td>GetSize()</td>
</tr>
<tr>
<td>genomic_regions pos</td>
<td>RunModifyPos()</td>
<td>ModifyPos()</td>
<td>ModifyPos()</td>
</tr>
<tr>
<td>genomic_regions reg</td>
<td>RunConvertToREG()</td>
<td>PrintREG()</td>
<td>---</td>
</tr>
<tr>
<td>genomic_regions rnd</td>
<td>RunRandomize()</td>
<td>Randomize()</td>
<td>---</td>
</tr>
<tr>
<td>genomic_regions select</td>
<td>RunSelect()</td>
<td>Select()</td>
<td>---</td>
</tr>
<tr>
<td>genomic_regions shift</td>
<td>RunShiftPos()</td>
<td>ShiftPos()</td>
<td>ShiftPos()</td>
</tr>
<tr>
<td>genomic_regions shiftp</td>
<td>RunShiftPos()</td>
<td>ShiftPos()</td>
<td>ShiftPos()</td>
</tr>
<tr>
<td>genomic_regions shuffle</td>
<td>RunShuffle()</td>
<td>RunShuffle()</td>
<td>---</td>
</tr>
<tr>
<td>genomic_regions sort</td>
<td>RunSort()</td>
<td>Sort()</td>
<td>---</td>
</tr>
<tr>
<td>genomic_regions split</td>
<td>RunSplit()</td>
<td>RunSplit()</td>
<td>---</td>
</tr>
<tr>
<td>genomic_regions strand</td>
<td>RunModifyStrand()</td>
<td>ModifyStrand()</td>
<td>not implemented yet</td>
</tr>
<tr>
<td>genomic_regions union</td>
<td>RunUnion()</td>
<td>Union()</td>
<td>---</td>
</tr>
<tr>
<td>genomic_regions wig</td>
<td>RunConvertToWIG()</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>genomic_regions win</td>
<td>RunSlidingWindows()</td>
<td>PrintWindows()</td>
<td>not implemented yet</td>
</tr>
<tr>
<td>genomic_regions x</td>
<td>RunExtractSeq()</td>
<td>PrintSeq()</td>
<td>---</td>
</tr>
<tr>
<td>genomic_regions gdist</td>
<td>RunGlobalCalcDistances()</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>genomic_regions inv</td>
<td>RunGlobalInvert()</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>genomic_regions link</td>
<td>RunGlobalLink()</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>genomic_regions test</td>
<td>RunGlobalTest()</td>
<td>---</td>
<td>---</td>
</tr>
</tbody>
</table>
2.3 The GenomicRegionSetScanner class

This class is used to scan by sliding windows an instance of GenomicRegionSet and is used in the genomic_scans command-line tool. The main advantage of this implementation of sliding window computations is that it is done sequentially without the need of storing the entire input intervals in memory. A practical use of this class is to develop customized window-based peak discovery algorithms. As shown in the example below, this class can be used to determine the number of reads in signal and control region sets in sliding windows along the entire genome:

```c++
#include "core.h"
#include "genomic_intervals.h"

// parameters
long int w = 500;  // window size
long int d = 25;   // window distance
bool verbose = true;
bool load_in_memory = false;

// initialize: create input region sets and associated scanners
map<string,long int> *bounds = ReadBounds(genome_file);
GenomicRegionSet signalReg(signal_file,10000,verbose,load_in_memory);
GenomicRegionSet controlReg(control_file,10000,verbose,load_in_memory);
GenomicRegionSetScanner signal_scanner(&signalReg,bounds,d,w,
false,false,'c');
GenomicRegionSetScanner control_scanner(&controlReg,bounds,d,w,false,false,'c');

// run: read window read counts sequentially in both files and test
while (true) {
    long int n = signal_scanner.Next();
    long int m = control_scanner.Next();
    if (n==-1) break;
    // ADD your statistical test HERE
}
```
2.4 The GenomicRegionSetOverlaps class and its extensions

This class is an abstract class used for determining and manipulating overlaps between two regions sets. It is extended into two classes.

SortedGenomicRegionSetOverlaps (renamed from GenomicRegionSetOverlapScanner in version 1.3) is used on sorted region sets. The sort order is first by chromosome, then (optionally) by strand, and finally by start position. The algorithm used to compute overlaps in this class is a generalization of the standard merge-sort algorithm modified so as to handle intervals. As before, the main advantage of this implementation is that processing is done sequentially without the need of storing the entire input intervals in memory.

The algorithm operates on sorted inputs, scans the files sequentially and computes all overlaps essentially using a merge-sort algorithm adapted to handle intervals. An intermediate buffer keeps all the overlaps of index regions with the current query, since they may also overlap with the next query.

INPUTS: query intervals Q and index intervals I.

PSEUDOCODE:

# Q and I are read sequentially as input streams
1. q = next(Q)    # read first query
2. i = next(I)    # read first interval
3. B = { }        # buffer for local overlaps
4. while (q) {
5.   if (q < i) q = next(Q)    # read next query
6.   else if (q > i) i = next(B∪I)    # read next interval
7.   else {
8.     B = { }    # buffer for local overlaps
9.     while (overlap(q,i)) {
10.    print q,i    # print overlaps
11.    B = B∪I    # store interval in buffer
12.    i = next(I)
13.   }    # store interval in buffer
14. }    # store interval in buffer
15. }

UnsortedGenomicRegionSetOverlaps is used on unsorted region sets. The algorithm used here is a modification of the algorithm proposed in [1], where we allow the number of levels and the number of bins per level to be chosen arbitrarily.

The example below demonstrates the use of both derived classes (this is taken from the source code file “genomic_overlaps.cpp”):

```
#include "core.h"
#include "genomic_intervals.h"

// open region sets
char *REF_REG_FILE = "exons.bed";
char *TEST_REG_FILE = "rnaseq.reads.bed";
```
GenomicRegionSet RefRegSet(REF_REG_FILE,BUFFER_SIZE,VERBOSE,true);
GenomicRegionSet TestRegSet(TEST_REG_FILE,BUFFER_SIZE,VERBOSE,false);

// process overlaps
GenomicRegionSetOverlaps *overlaps;
if (IS_SORTED) overlaps = new SortedGenomicRegionSetOverlaps(&TestRegSet,&RefRegSet,false);
else overlaps = new UnsortedGenomicRegionSetOverlaps(&TestRegSet,&RefRegSet);
unsigned long int *coverage;
coverage = overlaps->CalcIndexCoverage(MATCH_GAPS,IGNORE_STRAND,USE_VALUES);
Progress PRG("Printing densities...",RefRegSet.n_regions);
for (long int k=0; k<RefRegSet.n_regions; k++) {
    GenomicRegion *qreg = RefRegSet.R[k];
    long int qreg_size = MATCH_GAPS ? (qreg->I.back()->STOP-qreg->I.front()->START+1) :
                           qreg->GetSize();
    double density = (double)coverage[k]/qreg_size;
    if (density>=MIN_DENSITY) printf("%s\t%.4e\n", qreg->LABEL, density);
    PRG.Check();
}
PRG.Done();
delete coverage;
delete overlaps;
3 Installation

*GenomicTools* can be downloaded from the Google code repository [http://code.google.com/p/ibm-cbc-genomic-tools](http://code.google.com/p/ibm-cbc-genomic-tools). The tools have a dependency on the GNU Scientific Library (GSL) to compile. GSL can be downloaded from [http://www.gnu.org/software/gsl/](http://www.gnu.org/software/gsl/). To install GenomicTools, follow these simple instructions:

```bash
tar xvzf genomic-tools-VERSION-src+doc.tgz
cd genomic-tools
make
sudo cp bin/* /usr/local/bin
```
4 The GenomicTools suite

4.1 The genomic_regions tool

The genomic_regions tool is designed to allow the user to perform basic operations for manipulating genomic region files. These are: (a) line-based operations, such as shifting, shuffling, sorting, and modifying genomic regions, and (b) file-based operations such as linking, or inverting.

The tool’s function can be summarized in the following three steps:

• supply input region-set (REG, GFF, BED, SAM)
• sequentially process the input by interval or region depending on the operation
• output either another region-set (in the same format) or a “summary” (e.g. region sizes, distances)

In the following subsections, we summarize the basic functionality and options for each operation. This information is actually embedded into the genomic_regions command-line tool and can be obtained via the “--help” option, for example:

$ genomic_regions align --help

For the complete list of available operators, simply type:

$ genomic_regions

In fact, in this manner, the user can get the latest updated version of description and options for each operation. However, in the following subsections, we also give a few examples.

The summary for each operation, contains the following information:

• command-line usage
• brief description
• details: allowed input formats, operand types (interval, region, region-set, etc), and requirements at the region and region-set level (see definitions in section 1.3)
• options: descriptions and default values
### 4.1.1 genomic_regions align

**USAGE:**

```plaintext
genomic_regions align [OPTIONS] <REGION-SET>
```

**DESCRIPTION:**

Prints alignments of input sequences given the reference genome sequence.

**DETAILS:**

* Input formats: SAM
* Operands: region, sequence, reference sequence
* Region requirements: chromosome/strand-compatible, sorted, non-overlapping
* Region-set requirements: none

**OPTIONS:**

- `--help` help
- `-h` help
- `-v` verbose mode
- `-Q` FASTA file containing all chromosomes (overrides options -D and -q)
- `-D` chromosome DNA file and map directory
- `-q` sequence map file

**EXAMPLES**

```plaintext
$ samtools view reads.bam
read#1 89 chromosome_1 565373 255 8M3D3M4I7M * 0 0 TACCAAGCTCACACCTCTGAC
EEFFFFFFFFHHHHHHHHHHHHHHHH

$ cat genome.seq >chromosome_1
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN

$ samtools view reads.bam | genomic_regions align -Q genome.seq
>read#1 89 chromosome_1 565373 255 8M3D3M4I7M * 0 0 TACCAAGCTCACACCTCTGAC
EEFFFFFFFFHHHHHHHHHHHHHHHH
TACCAAG***CTCACACCTCTTGAC
TACCAAGGCCACCC***CTCTGAC
==============DDD=X=IIII=======
EEFFFFFF***FHHHHHHHHHHHHHH
```
4.1.2 genomic_regions bed

**USAGE:**
```
genomic_regions bed [OPTIONS] <REGION-SET>
```

**DESCRIPTION:**
Converts input regions into BED format.

**DETAILS:**
* Input formats: REG, GFF, BED, SAM
* Operand: region
* Region requirements: compatible, sorted, non-overlapping
* Region-set requirements: none

**OPTIONS:**
```
--help help [true]
-h help [true]
-v verbose mode [false]
-t title []
-c color []
-p browser position []
-chr convert chromosome names from ENSEMBL to UCSC [false]
```

**EXAMPLES**
```
$ cat hg19.exon.reg | head -2
ENSG00000223972:ENST00000456328:ENSE00002082992 1 + 11869 12227
ENSG00000223972:ENST00000450305:ENSE00001948541 1 + 12010 12057

$ cat hg19.exon.reg | genomic_regions bed | head -2
1 11868 12227 ENSG00000223972:ENST00000456328:ENSE00002082992 1000 +
1 12009 12057 ENSG00000223972:ENST00000450305:ENSE00001948541 1000 +

$ cat hg19.exon.reg | genomic_regions bed -chr -t "ENSEMBL known exons" | head -3
track name='ENSEMBL known exons' itemRgb=On visibility=1
chr1 11868 12227 ENSG00000223972:ENST00000456328:ENSE00002082992 1000 +
chr1 12009 12057 ENSG00000223972:ENST00000450305:ENSE00001948541 1000 +
```
4.1.3 genomic_regions bounds

**USAGE:**
```
    genomic_regions bounds [OPTIONS] <REGION-SET>
```

**DESCRIPTION:**
Trims start/stop positions given the chromosome bounds and removes invalid intervals.

**DETAILS:**
* Input formats: REG, GFF, BED, SAM
* Operand: interval
* Region requirements: none
* Region-set requirements: none

**OPTIONS:**
- **--help** help [true]
- **-h** help [true]
- **-v** verbose mode [false]
- **-g** genome region-set file []

---

**FUNCTION**

<table>
<thead>
<tr>
<th>Chromosome</th>
<th>==================================</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>----------------------</td>
</tr>
<tr>
<td>Output</td>
<td>----------------------</td>
</tr>
</tbody>
</table>

**EXAMPLES**

```
$ cat inp.bed
chr1 -21 249250625 geneA 1000 + -21 249250625 0 2 121,5 0,249250641
chr1 249999999 250001000 geneB 1000 +

$ cat hg19.genome.bed
chr1 0 249250621
chr10 0 135534747
chr11 0 135006516

$ cat inp.bed | genomic_regions bounds -g hg19.genome.bed
chr1 0 249250621 geneA 1000 + 0 2 100,1 0,249250620
```
4.1.4  genomic_regions center

USAGE:
genomic_regions center [OPTIONS] <REGION-SET>

DESCRIPTION:
Prints center interval.

DETAILS:
* Input formats: REG, GFF, BED, SAM
* Operand: interval
* Region requirements: none
* Region-set requirements: none

OPTIONS:
  --help    help                              [true]
  -h        help                              [true]
  -v        verbose mode                     [false]

FUNCTION

Input  ------------  =  =  =

Output  =  =  =  =

EXAMPLES

$ cat inp.reg
genA    1 1 100 1 520 800
geneB   1 2 102

$ cat inp.reg | genomic_regions center
genA    1 50 51 1 660 660
geneB   1 52 52
4.1.5  genomic_regions connect

**USAGE:**

    genomic_regions connect [OPTIONS] <REGION-SET>

**DESCRIPTION:**

Connects intervals from minimum start to maximum stop position.

**DETAILS:**

* Input formats: REG, GFF, BED, SAM
* Operand: region
* Region requirements: chromosome/strand-compatible
* Region-set requirements: none

**OPTIONS:**

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>-h</td>
<td>help</td>
<td>[true]</td>
</tr>
<tr>
<td>-v</td>
<td>verbose mode</td>
<td>[false]</td>
</tr>
</tbody>
</table>

**FUNCTION**

<table>
<thead>
<tr>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**EXAMPLES**

```
$ cat inp.reg
geneA  1 + 100 200 1 + 300 400 1 + 1000 1200
geneB  1 + 1000 1100 1 + 1090 1500

$ cat inp.reg | genomic_regions connect
geneA  1 + 100 1200
geneB  1 + 1000 1500
```
4.1.6 genomic_regions diff

**USAGE:**
genomic_regions diff [OPTIONS] <REGION-SET>

**DESCRIPTION:**
Computes the difference between successive intervals.

**DETAILS:**
* Input formats: REG, BED, SAM
* Operand: interval pair
* Region requirements: chromosome/strand-compatible, sorted, non-overlapping
* Region-set requirements: none

**OPTIONS:**
--help help [true]
-h help [true]
-v verbose mode [false]

---

**FUNCTION**

<table>
<thead>
<tr>
<th>Input</th>
<th>------</th>
<th>-----</th>
<th>------</th>
<th>-----</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output</td>
<td>------</td>
<td>------</td>
<td>-------</td>
<td>----</td>
</tr>
</tbody>
</table>

---

**EXAMPLES**

$ cat inp.reg
geneA 1 + 100 200 1 + 300 400 1 + 1000 1200
geneB 1 + 1000 1100 1 + 1190 1500

$ cat inp.reg | genomic_regions diff
geneA 1 + 201 299 1 + 401 999
geneB 1 + 1101 1189
4.1.7  genomic_regions dist

**USAGE:**
genomic_regions dist [OPTIONS] <REGION-SET>

**DESCRIPTION:**
Computes distances between pairs of successive intervals.

**DETAILS:**
* Input formats: REG, BED, SAM
* Operand: interval pair
* Region requirements: chromosome/strand-compatible
* Region-set requirements: none

**OPTIONS:**
--help help                                 [true]
h     help                                   [true]
v     verbose mode                           [false]
op1  reference point of 1st interval in pair (1=start, 2=stop, 5p=5'-end, 3p=3'-end)  [1]
op2  reference point of 2nd interval in pair (1=start, 2=stop, 5p=5'-end, 3p=3'-end)  [1]

**FUNCTION**

Input  ===== ← d₁ →  =============== ← d₂ →  ===============

Output (op1='2',op2='1')  d₁ d₂

**EXAMPLES**

```
$ cat inp.reg
 geneA  1   +100 200 1   +300 400
 geneB  1   +1000 1100 1   +1190 1500

$ cat inp.reg | genomic_regions dist
 geneA  200
 geneB  190
```
4.1.8  genomic_regions divide

USAGE:
genomic_regions divide [OPTIONS] <REGION-SET>

DESCRIPTION:
Divides intervals in the middle.

DETAILS:
* Input formats: REG, BED
* Operand: interval
* Region requirements: none
* Region-set requirements: none

OPTIONS:
--help  help  [true]
-h      help  [true]
-v      verbose mode  [false]

FUNCTION

Input

Output

EXAMPLES

$ cat inp.reg
geneA  1 + 100 200 1 + 300 400 1 + 1000 1200
geneB  1 + 1000 1100 1 + 1190 1500

$ cat inp.reg | genomic_regions divide
geneA  1 + 100 149 1 + 150 200 1 + 300 349 1 + 350 400 1 + 1000 1099 1 + 1100 1200
geneB  1 + 1000 1049 1 + 1050 1100 1 + 1190 1344 1 + 1345 1500
4.1.9  genomic_regions fix

USAGE:
    genomic_regions fix [OPTIONS] <REGION-SET>

DESCRIPTION:
    Removes invalid intervals, i.e. start<1 or start>stop.

DETAILS:
    * Input formats: REG, GFF, BED, SAM
    * Operand: interval
    * Region requirements: none
    * Region-set requirements: none

OPTIONS:
    --help help [true]
    -h help [true]
    -v verbose mode [false]

EXAMPLES

$ cat x
invalidA 1 -20 100
invalidB 1 +100 20
goodA 1 +1 1
goodB 1 +200 250

$ cat x | genomic_regions fix
goodA 1 +1 1
goodB 1 +200 250
4.1.10 genomic_regions gdist

**USAGE:**
```bash
genomic_regions gdist [OPTIONS] <REGION-SET>
```

**DESCRIPTION:**
Computes distances of successive regions.

**DETAILS:**
* Input formats: REG, GFF, BED, SAM
* Operand: region-pair
* Region requirements: single-interval
* Region-set requirements: sorted by chromosome/strand/start

**OPTIONS:**
- `--help` help               [true]
- `-h` help                   [true]
- `-v` verbose mode          [false]
- `-op1` reference point of 1st interval in pair (1=start, 2=stop, 5p=5'-end, 3p=3'-end) [1]
- `-op2` reference point of 2nd interval in pair (1=start, 2=stop, 5p=5'-end, 3p=3'-end) [1]

**EXAMPLES**

$ cat inp.reg
  geneA  1 + 100 1100
  geneB  1 + 1000 1500

$ cat inp.reg | genomic_regions gdist
genomeA  geneB  900
4.1.11 `genomic_regions int`

<table>
<thead>
<tr>
<th>USAGE:</th>
<th>genomic_regions int [OPTIONS] &lt;REGION-SET&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>DESCRIPTION:</td>
<td>Computes the intersection of input intervals.</td>
</tr>
<tr>
<td>DETAILS:</td>
<td>* Input formats: REG</td>
</tr>
<tr>
<td></td>
<td>* Operand: region</td>
</tr>
<tr>
<td></td>
<td>* Region requirements: chromosome/strand-compatible</td>
</tr>
<tr>
<td></td>
<td>* Region-set requirements: none</td>
</tr>
<tr>
<td>OPTIONS:</td>
<td>--help help [true]</td>
</tr>
<tr>
<td></td>
<td>-h help [true]</td>
</tr>
<tr>
<td></td>
<td>-v verbose mode [false]</td>
</tr>
</tbody>
</table>

**FUNCTION**

```
Input
+---+---+---+---+---+
|   |   |   |   |   |
Output
|   |   |
```

**EXAMPLES**

```
$ cat inp.reg
geneA 1 + 100 200 1 + 150 400
geneB 1 + 1000 1100 1 + 1000 1200 1 + 1090 1500

$ cat inp.reg | genomic_regions int
geneA 1 + 150 200
geneB 1 + 1090 1100
```
4.1.12 genomic_regions inv

**USAGE:**
```
genomic_regions inv [OPTIONS] <REGION-SET>
```

**DESCRIPTION:**
Inverts regions given the genome chromosomal boundaries.

**DETAILS:**
* Input formats: REG, GFF, BED, SAM
* Operand: region-set
* Region requirements: single-interval
* Region-set requirements: sorted by chromosome/strand/start

**OPTIONS:**

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>--help</td>
<td>help</td>
<td>[true]</td>
</tr>
<tr>
<td>-h</td>
<td>help</td>
<td>[true]</td>
</tr>
<tr>
<td>-v</td>
<td>verbose mode</td>
<td>[false]</td>
</tr>
<tr>
<td>-g</td>
<td>genome region-set file</td>
<td>[]</td>
</tr>
</tbody>
</table>

**FUNCTION**

<table>
<thead>
<tr>
<th>Chromosome</th>
<th>-------------------------------------------------</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>==================================================</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input</th>
<th>-------------------------------------------------</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>------- ------- ------- ------- ------- -------</td>
</tr>
<tr>
<td></td>
<td>------- ------- ------- ------- ------- -------</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Output</th>
<th>------- ------- ------- ------- ------- -------</th>
</tr>
</thead>
</table>

**EXAMPLES**

```
$ cat inp.reg
geneA  1 + 100 1100
geneB  1 + 1000 1500

$ head -2 genome.reg
chromosome_1  1 + 1 249250621
chromosome_1  1 - 1 249250621

$ genomic_regions inv inp.reg
_  1 + 1 99
_  1 + 1501 249250621
```
4.1.13 genomic_regions link

**USAGE:**
genomic_regions link [OPTIONS] <REGION-SET>

**DESCRIPTION:**
Links consecutive regions to produce a non-overlapping set.

**DETAILS:**
* Input formats: REG, GFF, BED, SAM
* Operand: region-set
* Region requirements: single-interval
* Region-set requirements: sorted by chromosome/(strand)/start

**OPTIONS:**
- `--help` help [true]
- `-h` help [true]
- `-v` verbose mode [false]
- `-s` input regions are sorted by strand [false]
- `-d` maximum difference between successive regions [0]

---

**FUNCTION**

```
Input

Output
```

---

**EXAMPLES**

```
$ cat inp.reg
geneA  1 + 100 1100
geneB  1 + 1000 1500

$ cat inp.reg | genomic_regions -link
geneA,geneB  1 + 100 1500
```
4.1.14 genomic_regions n

**USAGE:**
```
   genomic_regions n [OPTIONS] <REGION-SET>
```

**DESCRIPTION:**
Computes sum of lengths of intervals in a region (including possible overlaps).

**DETAILS:**
* Input formats: REG, GFF, BED, SAM
* Operand: region
* Region requirements: none
* Region-set requirements: none

**OPTIONS:**
```
--help    help [true]
-h        help [true]
-v        verbose mode [false]
```

**EXAMPLES**
```
$ cat inp.reg
  geneA  1 + 100 200 1 + 150 400
  geneB  1 + 1000 1100 1 + 1000 1200 1 + 1090 1500

$ cat inp.reg | genomic_regions n
  geneA  352
  geneB  713
```
4.1.15 genomic_regions pos

**USAGE:**
genomic_regions pos [OPTIONS] <REGION-SET>

**DESCRIPTION:**
Modifies interval start/stop positions.

**DETAILS:**
* Input formats: REG, GFF, BED, SAM
* Operand: interval
* Region requirements: none
* Region-set requirements: none

**OPTIONS:**
- --help       help                     [true]
  -h           help                     [true]
  -v           verbose mode            [false]
  -op          position operation (1=start, 2=stop, 5p=5'-end, 3p=3'-end, c=center) [1]
  -c           position shift           [0]

**FUNCTION**

<table>
<thead>
<tr>
<th>Input</th>
<th>++++++++++++++++++</th>
<th>(fwd strand)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>-----------------</td>
<td>--------------</td>
</tr>
<tr>
<td>Output (-op 1)</td>
<td>+++++++</td>
<td>-------</td>
</tr>
<tr>
<td>Output (-op 2)</td>
<td>+++++++</td>
<td>-------</td>
</tr>
<tr>
<td>Output (-op 5p)</td>
<td>+++++++</td>
<td>-------</td>
</tr>
<tr>
<td>Output (-op 3p)</td>
<td>+++++++</td>
<td>-------</td>
</tr>
</tbody>
</table>

**EXAMPLES**

$ cat sample.reg
geneA  1 + 100 200 1 + 300 400 1 + 1000 1200
geneB  1 + 1000 1100 1 + 1190 1500

$ genomic_regions pos -op 3p -c 10 sample.reg
geneA  1 + 190 200 1 + 390 400 1 + 1190 1200
geneB  1 + 1090 1100 1 + 1490 1500
4.1.16 genomic_regions reg

USAGE:
  genomic_regions reg [OPTIONS] <REGION-SET>

DESCRIPTION:
  Converts to REG format.

DETAILS:
  * Input formats: REG, GFF, BED, SAM
  * Operand: interval
  * Region requirements: chromosome/strand-compatible if option -c is set
  * Region-set requirements: none

OPTIONS:
  --help          help                         [true]
  -h              help                         [true]
  -v              verbose mode                [false]
  -c              print in compact starts/ends format [false]

EXAMPLES

$ cat inp.bed
chr1 99 1200 geneA 1000 + 99 1200 0 3 101,101,201 0,200,900
chr1 999 1500 geneB 1000 + 999 1500 0 2 101,311 0,190

$ cat inp.bed | genomic_regions reg
geneA  chr1 + 100 200 chr1 + 300 400 chr1 + 1000 1200
geneB  chr1 + 1000 1100 chr1 + 1190 1500
4.1.17 genomic_regions rnd

USAGE:
    genomic_regions rnd [OPTIONS] <REGION-SET>
DESCRIPTION:
    Randomizes region across entire genome (relative interval distances are preserved).
DETAILS:
    * Input formats: REG, GFF, BED, SAM
    * Operand: region
    * Region requirements: chromosome/strand-compatible, sorted, non-overlapping
    * Region-set requirements: none
OPTIONS:
    --help         help               [true]
    -h             help               [true]
    -v             verbose mode      [false]
    -g             genome region-set file []

FUNCTION

Chromosome
--------------

Region
-------

Output
------

EXAMPLES

$ cat inp.bed
chr1 0 800 geneA 1000 + 0 800 0 2 100,281 0,519
chr1 1 102 geneB 1000 +

$ cat genome.bed | head -2
chr1 0 249250621

$ cat inp.bed | genomic_regions rnd -g genome.bed
chr1 171844041 171844841 geneA 1000 + 171844041 171844841 0 2 100,281 0,519
chr1 146448932 146449033 geneB 1000 +
4.1.18 genomic_regions select

**USAGE:**
```
genomic_regions select [OPTIONS] <REGION-SET>
```

**DESCRIPTION:**
Selects a subset of intervals according to their position in the region.

**DETAILS:**
* Input formats: REG, BED, SAM
* Operand: region
* Region requirements: chromosome/strand-compatible
* Region-set requirements: none

**OPTIONS:**
```
-h  --help            help                         [true]
-h  --help            help                         [true]
-v  --verbose         verbose mode                [false]
-first  --first       select the first interval  [false]
-last   --last         select the last interval   [false]
-5p     --5p           select the first from the 5' end  [false]
-3p     --3p           select the first from the 3' end  [false]
```

**EXAMPLES**

```
$ cat inp.bed
chr1 0 800 geneA 1000 - 0 800 0 2 100,281 0,519
chr1 1 102 geneB 1000 +

$ cat inp.bed | genomic_regions select -first
chr1 0 100 geneA 1000 - 0 100 0 1 100 0
chr1 1 102 geneB 1000 +

$ cat inp.bed | genomic_regions select -5p
chr1 519 800 geneA 1000 - 519 800 0 1 281 0
chr1 1 102 geneB 1000 +
```
4.1.19 genomic_regions shift

Usage:
genomic_regions shift [OPTIONS] <REGION-SET>

Description:
Shifts interval start/stop positions.

Details:
* Input formats: REG, GFF, BED, SAM
* Operand: interval
* Region requirements: none
* Region-set requirements: none

Options:
--help help [true]  -h help [true]  -v verbose mode [false]  -start shift distance for start position [0]  -stop shift distance for stop position [0]

Function

Input

\[ \text{start} \rightarrow \text{stop} \]

Output

Examples

$ cat inp.reg
geneA  1 + 100 200 1 + 300 400 1 + 1000 1200
geneB  1 + 1000 1100 1 + 1190 1500

$ cat inp.reg | genomic_regions -shift -1 10 -2 5
geneA  1 + 110 205 1 + 310 405 1 + 1010 1205
geneB  1 + 1010 1105 1 + 1200 1505
4.1.20 genomic_regions shiftp

**USAGE:**
```
genomic_regions shiftp [OPTIONS] <REGION-SET>
```

**DESCRIPTION:**
Shifts interval 5'/3' positions.

**DETAILS:**
- * Input formats: REG, GFF, BED, SAM
- * Operand: interval
- * Region requirements: none
- * Region-set requirements: none

**OPTIONS:**
- `-help` help [true]
- `-h` help [true]
- `-v` verbose mode [false]
- `-5p` shift distance for 5' end [0]
- `-3p` shift distance for 3' end [0]

**FUNCTION**
```
Input --------------------------------- (rev strand)

  3p  

Output ---------------------------------

  5p  
```

**EXAMPLES**
```
$ cat inp.reg
  geneA  1 + 100 200 1 + 1000 2000
  geneB  1 - 1500 2500

$ cat inp.reg | genomic_regions shiftp -5p -10 -3p +100
  geneA  1 + 90 300 1 + 990 2100
  geneB  1 - 1400 2510
```
4.1.21 genomic_regions shuffle

**USAGE:**
```
  genomic_regions shuffle [OPTIONS] <REGION-SET>
```

**DESCRIPTION:**
Shuffles intervals within given reference region-set.

**DETAILS:**
- Input formats: REG, GFF, BED, SAM
- Operand: interval
- Region requirements: single-interval
- Region-set requirements: none

**OPTIONS:**
- `-help` `--help` help [true]
- `-h` help [true]
- `-v` verbose mode [false]
- `-r` reference region file []

---

**FUNCTION**

<table>
<thead>
<tr>
<th>Reference</th>
<th>-----------------</th>
<th>-----------------</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>AAAAAAA</td>
<td>BBBB   CCC</td>
</tr>
<tr>
<td>Output</td>
<td>BBBB</td>
<td>AAAAAAA      CCC</td>
</tr>
</tbody>
</table>

---

**EXAMPLES**

```bash
$ cat inp.reg
geneA  1 + 1 100
geneB  1 + 1000 1100

$ cat reference.reg
ref_1  1 + 1 1000
ref_2  1 + 7000 10000

$ cat inp.reg | genomic_regions shuffle -r reference.reg
geneA  1 + 889 988
geneB  1 + 7795 7895
```
### 4.1.22 genomic_regions sort

**Usage:**

```
genomic_regions sort [OPTIONS] <REGION-SET>
```

**Description:**

Sorts region intervals by start position.

**Details:**

- Input formats: REG
- Operand: region
- Region requirements: chromosome/strand-compatible
- Region-set requirements: none

**Options:**

```
--help       help [true]
-h           help [true]
-v           verbose mode [false]
```

## Function

<table>
<thead>
<tr>
<th>Input</th>
<th>222222</th>
<th>4444</th>
<th>3333333333</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>111111</td>
</tr>
<tr>
<td>Output</td>
<td>111111</td>
<td>222</td>
<td>4444444444</td>
</tr>
<tr>
<td></td>
<td></td>
<td>33333</td>
<td></td>
</tr>
</tbody>
</table>

## Examples

```bash
$ cat inp.reg
geneA  1 - 100 200 1 + 1100 1400 1 + 1000 1200
geneB  1 + 1000 1100 1 + 1190 1500

$ cat inp.reg | genomic_regions sort
geneA  1 + 100 200 1 + 1000 1200 1 + 1100 1400
geneB  1 + 1000 1100 1 + 1190 1500
```
4.1.23 genomic_regions split

**USAGE:**
```
genomic_regions split [OPTIONS] <REGION-SET>
```

**DESCRIPTION:**
Splits regions into their intervals which are printed on separate lines.

**DETAILS:**
- Input formats: REG, GFF, BED, SAM
- Operand: region
- Region requirements: none
- Region-set requirements: none

**OPTIONS:**
- `--help` help [true]
- `-h` help [true]
- `-v` verbose mode [false]

**EXAMPLES**

```
$ cat inp.reg
geneA  1 + 100 200
geneA  1 + 300 400
geneA  1 + 1000 1200

geneB  1 + 1000 1100
geneB  1 + 1190 1500

$ cat inp.reg | genomic_regions split
geneA  1 + 100 200
geneA  1 + 300 400
geneA  1 + 1000 1200
geneB  1 + 1000 1100
geneB  1 + 1190 1500
```
4.1.24  genomic_regions strand

| USAGE: | genomic_regions strand [OPTIONS] <REGION-SET> |
| DESCRIPTION: | Modifies interval strand information. |
| DETAILS: | * Input formats: REG, GFF, BED, SAM |
| | * Operand: interval |
| | * Region requirements: chromosome/strand-compatible |
| | * Region-set requirements: none |
| OPTIONS: | --help  help  [true] |
| | -h  help  [true] |
| | -v  verbose mode  [false] |
| | -op  strand operation (+=positive, -=negative, r=reverse)  [+] |
| | -s  returns a sorted region set (only works if input is sorted)  [false] |

**EXAMPLES**

$ cat inp.reg
geneA  1 + 100 200 1 + 300 400 1 + 1000 1200
geneB  1 + 1000 1100 1 + 1190 1500

$ cat inp.reg | genomic_regions strand --op r
geneA  1 - 100 200 1 - 300 400 1 - 1000 1200
geneB  1 - 1000 1100 1 - 1190 1500
### 4.1.25 genomic_regions test

**USAGE:**
```
genomic_regions test [OPTIONS] <REGION-SET>
```

**DESCRIPTION:**
Tests whether genomic regions are sorted and non-overlapping.

**DETAILS:**
- Input formats: REG, GFF, BED, SAM
- Operand: region
- Region requirements: chromosome/strand-compatible, sorted, non-overlapping
- Region-set requirements: sorted by chromosome/(strand)/start

**OPTIONS:**
- `-h` or `--help` [true]
- `-v` or `--verbose` [false]
- `-s` or `--input-regions-are-sorted-by-strand` [false]

**EXAMPLES**

Example: geneC is contained within geneB, and geneA overlaps geneB

```bash
$ cat inp.reg
geneA  1 + 100 1100
geneB  1 + 1000 1500
geneC  1 + 1150 1151
```

```bash
$ cat inp.reg | genomic_regions test
* The file is sorted! Found 1 inclusions and 1 overlaps.
```
### 4.1.26 genomic_regions union

**Usage:**
```
    genomic_regions union [OPTIONS] <REGION-SET>
```

**Description:**
Computes union of region intervals.

**Details:**
- Input formats: REG, GFF, BED, SAM
- Operand: region
- Region requirements: chromosome/strand-compatible
- Region-set requirements: none

**Options:**
- `--help` help [true]
- `-h` help [true]
- `-v` verbose mode [false]

---

**Function**

<table>
<thead>
<tr>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>-----</td>
<td>---------------------</td>
</tr>
<tr>
<td>-----</td>
<td></td>
</tr>
<tr>
<td>-----</td>
<td></td>
</tr>
<tr>
<td>-----</td>
<td></td>
</tr>
<tr>
<td>-----</td>
<td></td>
</tr>
<tr>
<td>-----</td>
<td></td>
</tr>
<tr>
<td>-----</td>
<td></td>
</tr>
<tr>
<td>-----</td>
<td></td>
</tr>
<tr>
<td>-----</td>
<td></td>
</tr>
<tr>
<td>-----</td>
<td></td>
</tr>
<tr>
<td>-----</td>
<td></td>
</tr>
<tr>
<td>-----</td>
<td></td>
</tr>
<tr>
<td>-----</td>
<td></td>
</tr>
</tbody>
</table>

---

**Examples**

```
$ cat inp.reg
geneA  1 + 1 100 1 + 50 200
geneB  2 - 50 100 2 - 70 110

$ cat inp.reg | genomic_regions union
geneA  1 + 1 200
geneB  2 - 50 110
```
4.1.27 genomic_regions wig

**USAGE:**
genomic_regions wig [OPTIONS] <REGION-SET>

**DESCRIPTION:**
Converts to UCSC wiggle format.

**DETAILS:**
* Input formats: REG, GFF, BED, SAM
* Operand: region
* Region requirements: single-interval, forward-strand
* Region-set requirements: sorted by chromosome and start position

**OPTIONS:**

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>--help</td>
<td>help</td>
<td>[true]</td>
</tr>
<tr>
<td>-h</td>
<td>help</td>
<td>[true]</td>
</tr>
<tr>
<td>-v</td>
<td>verbose mode</td>
<td>[false]</td>
</tr>
<tr>
<td>-t</td>
<td>title</td>
<td>[]</td>
</tr>
<tr>
<td>-c</td>
<td>color</td>
<td>[200,0,0]</td>
</tr>
<tr>
<td>-s</td>
<td>span</td>
<td>[1]</td>
</tr>
<tr>
<td>-p</td>
<td>browser position</td>
<td>[]</td>
</tr>
<tr>
<td>-o</td>
<td>track type options</td>
<td>[]</td>
</tr>
<tr>
<td>-chr</td>
<td>convert chromosome names from ENSEMBL to UCSC</td>
<td>[false]</td>
</tr>
</tbody>
</table>

**EXAMPLES**

$ cat inp.reg
42 1 + 1 100
102 2 + 50 100

$ cat inp.reg | genomic_regions wig --s 10 --t example
variableStep chrom=1 span=10
10034510 42
variableStep chrom=2 span=10
50 102
4.1.28  genomic_regions win

**USAGE:**
```
genomic_regions win [OPTIONS] <REGION-SET>
```
**DESCRIPTION:**
Creates new intervals by sliding windows.
**DETAILS:**
* Input formats: REG, BED
* Operand: region
* Region requirements: single-interval
* Region-set requirements: none
**OPTIONS:**
```
--help             help                          [true]
-h                 help                          [true]
-v                 verbose mode                [false]
-s                 window size                  [1]
-d                 window distance              [1]
```

**FUNCTION**

Input
```
|---|---|
```
Output
```
|---|
```

**EXAMPLES**

```
$ cat sample.reg
geneA 1 + 1 100
geneB 2 + 50 100

$ cat sample.reg | genomic_regions win -d 10 -s 20
geneA#1 1 + 1 20
geneA#2 1 + 11 30
geneA#3 1 + 21 40
...
```
4.1.29 genomic_regions x

**USAGE:**
genomic_regions x [OPTIONS] <REGION-SET>

**DESCRIPTION:**
Extracts region sequence from DNA.

**DETAILS:**
* Input formats: REG, GFF, BED, SAM
* Operand: region
* Region requirements: chromosome/strand-compatible
* Region-set requirements: none

**OPTIONS:**
--help help [true]
-h help [true]
-v verbose mode [false]
-Q FASTA file containing all chromosomes (overrides options -D and -q) []
-D chromosome DNA file and map directory []
-q sequence map file [chromosome.map]
-r replace 'N' characters with 'a' [false]
-i ignore boundary errors [false]

**EXAMPLES**

```
$ cat inp.reg
geneA 1 + 10034510 10034550
geneB 2 + 50 100

$ head -2 chromosome.map
1 chromosome.1.fa.dna
10 chromosome.10.fa.dna

$ cat inp.reg | genomic_regions x
geneA 1 + 10034510 10034550
GATTCAGCTCACTGCAAACTCCGCCTCCCAGGCTCACACC
geneB 2 + 50 100
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
```
4.2 The genomic_overlaps tool

The genomic_overlaps tool allows the user to compute various measures of overlaps between sets of regions. This is achieved by providing a set of operations and a variety of options for: (a) finding regions that match or partially overlap, (b) counting the number of matches, (c) calculating densities of matched regions, and (d) computing relative distances (i.e. offsets) between overlapping regions. Applications include computation of gene expression, construction of average read profiles across gene bodies, transcriptional start sites (TSSs) etc, and enrichment analyses of virtually any genomic dataset, such as genes of specific functional categories, repeat types, SNPs, cancer-associated regions.

The tool’s function can be summarized in the following three steps:

• supply input test and reference region-sets (REG, GFF, BED, SAM)
• perform pair-wise overlap operation between each test and reference region
• output either another region-set (in the same format) or a “summary” (e.g. densities, offsets)

Some key common options:

• -gaps: when a region comprises multiple non-overlapping intervals, then “gaps” are created in-between; if this option is not set (default behavior), then two regions (from test and reference region-sets) are considered to overlap only if there is some overlap between the actual intervals; in contrast, if this option is set, then an overlap is declared even if the entire overlapping segment is in the gaps; for example, if a region represents the exons of a gene, -gaps=true will report overlaps with the entire gene body, whereas -gaps=false will only report overlaps with the actual exons.

• -S: this option can be set if both input region-sets are sorted by chromosome and start position (if -s is also set, then the sort order is determined by chromosome, then strand, and finally start position); in such a scenario, a faster algorithm (described in section 2.4) will be used to perform the overlap operations; if this option is not set, then the standard binning algorithm is used.

• -i: if this option is set, then overlaps are reported without considering the strand (default behavior reports only strand-specific overlaps).

In the following subsections, we summarize the basic functionality and options for each operation. As described previously for the genomic_regions tool, this information is actually embedded into the genomic_overlaps command-line tool and can be obtained via the “--help” option.
4.2.1  genomic_overlaps count

USAGE:
genomic_overlaps count [OPTIONS] REFERENCE-REGION-FILE <TEST-REGION-FILE>

DESCRIPTION:
Counts the number of overlapping test regions per reference region.

DETAILS:
* Input formats: REG, GFF, BED, SAM
* Operands: region, region-set
* Region requirements: chromosome/strand-compatible, sorted, non-overlapping
* Region-set requirements: none

OPTIONS:
--help  help  [true]
-h      help  [true]
-v      verbose mode  [false]
-S      test and reference regions are sorted by chromosome and start position  [false]
-s      test and reference regions are also sorted by strand (-S must be set)  [false]
-i      ignore strand while finding overlaps  [false]
-gaps  matching gaps between intervals are considered overlaps  [false]
-val    use values contained in the labels of index intervals  [false]
-min    minimum count  [0]

EXAMPLES

$ cat exon.bed
chr1 95 150 exonA 255 +

$ cat reads.bed
chr1 100 149 read#1
chr1 101 150 read#2
chr1 101 150 read#3
chr1 101 150 read#4
chr1 105 154 read#5
chr1 200 249 read#6

$ cat reads.bed | genomic_overlaps count exon.bed
geneA  5
4.2.2  genomic_overlaps coverage

USAGE:
genomic_overlaps coverage [OPTIONS] REFERENCE-REGION-FILE <TEST-REGION-FILE>

DESCRIPTION:
Calculates the depth coverage (i.e. the total number of overlapping nucleotides) per reference region.

DETAILS:
* Input formats: REG, GFF, BED, SAM
* Operands: region, region-set
* Region requirements: chromosome/strand-compatible, sorted, non-overlapping
* Region-set requirements: none

OPTIONS:
--help        help [true]
-h            help [true]
-v            verbose mode [false]
-S            test and reference regions are sorted by chromosome and start position [false]
-s            test and reference regions are also sorted by strand (-S must be set) [false]
-i            ignore strand while finding overlaps [false]
-gaps         matching gaps between intervals are considered overlaps [false]
-val          use values contained in the labels of index intervals [false]
-min          minimum coverage [0]

EXAMPLES

$ cat exon.bed
chr1 95 150 exonA 255 +

$ cat reads.bed
chr1 100 149 read#1
chr1 101 150 read#2
chr1 101 150 read#3
chr1 101 150 read#4
chr1 105 154 read#5
chr1 200 249 read#6

$ cat reads.bed | genomic_overlaps coverage exon.bed
geneA  241
4.2.3  genomic_overlaps density

USAGE:
    genomic_overlaps density [OPTIONS] REFERENCE-REGION-FILE <TEST-REGION-FILE>

DESCRIPTION:
    Computes the density (i.e. the coverage divided by the size of the reference region) of overlaps per reference region.

DETAILS:
    * Input formats: REG, GFF, BED, SAM
    * Operands: region, region-set
    * Region requirements: chromosome/strand-compatible, sorted, non-overlapping
    * Region-set requirements: none

OPTIONS:
    --help  help              [true]
    -h      help              [true]
    -v      verbose mode      [false]
    -S      test and reference regions are sorted by chromosome and start position [false]
    -s      test and reference regions are also sorted by strand (-S must be set) [false]
    -i      ignore strand while finding overlaps [false]
    -gaps   matching gaps between intervals are considered overlaps [false]
    -val    use values contained in the labels of index intervals [false]
    -min    minimum density   [0.000000]

EXAMPLES

$ cat exon.bed
chr1 95 150 exonA 255 +

$ cat reads.bed
chr1 100 149 read#1
chr1 101 150 read#2
chr1 101 150 read#3
chr1 101 150 read#4
chr1 105 154 read#5
chr1 200 249 read#6

$ cat reads.bed | genomic_overlaps density exon.bed
geneA  4.3818e+00
4.2.4  genomic_overlaps intersect

**USAGE:**
```
    genomic_overlaps intersect [OPTIONS] REFERENCE-REGION-FILE <TEST-REGION-FILE>
```

**DESCRIPTION:**
Computes the intersection between all pairs of test and reference regions. Results are grouped by test region.

**DETAILS:**
* Input formats: REG, GFF, BED, SAM
* Operands: region, region-set
* Region requirements: chromosome/strand-compatible, sorted, non-overlapping
* Region-set requirements: none

**OPTIONS:**
```
--help help          [true]
-h help             [true]
-v verbose mode     [false]
-S test and reference regions are sorted by chromosome and start position [false]
-s test and reference regions are also sorted by strand (-S must be set) [false]
-i ignore strand while finding overlaps [false]
-label print query label for each match [false]
```

**EXAMPLES**

```
$ cat exon.bed
chr1 95 150 exonA 255 +

$ cat reads.bed
chr1 100 149 read#1
chr1 101 150 read#2
chr1 101 150 read#3
chr1 101 150 read#4
chr1 105 154 read#5
chr1 200 249 read#6

$ cat reads.bed | genomic_overlaps intersect -label exon.bed
chr1 100 149 read#1:geneA
chr1 101 150 read#2:geneA
chr1 101 150 read#3:geneA
chr1 101 150 read#4:geneA
chr1 105 154 read#5:geneA
```
4.2.5  genomic_overlaps offset

USAGE:
genomic_overlaps offset [OPTIONS] REFERENCE-REGION-FILE <TEST-REGION-FILE>

DESCRIPTION:
Computes the distances of test regions from their overlapping reference regions.

DETAILS:
* Input formats: REG, GFF, BED, SAM
* Operands: region, region-set
* Region requirements: chromosome/strand-compatible, sorted, non-overlapping
* Region-set requirements: none

OPTIONS:
--help  help                          [true]
-h      help                          [true]
-v      verbose mode                  [false]
-S      test and reference regions are sorted by chromosome and start position  [false]
-s      test and reference regions are also sorted by strand (-S must be set)  [false]
-i      ignore strand while finding overlaps  [false]
-gaps  matching gaps between intervals are considered overlaps  [false]
-label  print test region labels  [false]
-op     reference point (1=start, 2=stop, 5p=5'-end, 3p=3'-end)  [1]
-a      print distances as a fraction of total size  [false]
-c      print center of interval only  [false]

EXAMPLES

$ cat exon.bed
chr1 95 150 exonA 255 +

$ cat reads.bed
chr1 100 149 read#1
chr1 101 150 read#2
chr1 101 150 read#3
chr1 101 150 read#4
chr1 105 154 read#5
chr1 200 249 read#6

$ cat reads.bed | genomic_overlaps offset -label exon.bed
geneA  read#1  5  53
geneA  read#2  6  54
geneA  read#3  6  54
geneA  read#4  6  54
geneA  read#5 10  58
4.2.6  genomic_overlaps overlap

**USAGE:**

```bash
genomic_overlaps overlap [OPTIONS] REFERENCE-REGION-FILE <TEST-REGION-FILE>
```

**DESCRIPTION:**

Finds the overlaps between all pairs of test and reference regions. Results are grouped by test region.

**DETAILS:**

* Input formats: REG, GFF, BED, SAM
* Operands: region, region-set
* Region requirements: chromosome/strand-compatible, sorted, non-overlapping
* Region-set requirements: none

**OPTIONS:**

```bash
--help  help [true]
-h     help [true]
-v     verbose mode [false]
-S     test and reference regions are sorted by chromosome and start position [false]
-s     test and reference regions are also sorted by strand (-S must be set) [false]
-i     ignore strand while finding overlaps [false]
-gaps matching gaps between intervals are considered overlaps [false]
-label print query label for each match [false]
```

**EXAMPLES**

```bash
$ cat exon.bed
chr1 95 150 exonA 255 +

$ cat reads.bed
chr1 100 149 read#1
chr1 101 150 read#2
chr1 101 150 read#3
chr1 101 150 read#4
chr1 105 154 read#5
chr1 200 249 read#6

$ cat reads.bed | genomic_overlaps overlap -label exon.bed
chr1 100 149 read#1:geneA
chr1 101 150 read#2:geneA
chr1 101 150 read#3:geneA
chr1 101 150 read#4:geneA
chr1 105 154 read#5:geneA
```
4.2.7  genomic_overlaps subset

**USAGE:**

```
genomic_overlaps subset [OPTIONS] REFERENCE-REGION-FILE <TEST-REGION-FILE>
```

**DESCRIPTION:**

Picks a subset of test regions depending on their overlap with reference regions. Results are grouped by test region.

**DETAILS:**

* Input formats: REG, GFF, BED, SAM
* Operands: region, region-set
* Region requirements: chromosome/strand-compatible, sorted, non-overlapping
* Region-set requirements: none

**OPTIONS:**

```
--help help [true]
-h help [true]
-v verbose mode [false]
-S test and reference regions are sorted by chromosome and start position [false]
-s test and reference regions are also sorted by strand (-S must be set) [false]
-i ignore strand while finding overlaps [false]
-gaps matching gaps between intervals are considered overlaps [false]
-inv print test regions that do *not* overlap with reference regions [false]
```

**EXAMPLES**

```
$ cat exon.bed
chr1 95 150 exonA 255 +

$ cat reads.bed
chr1 100 149 read#1
chr1 101 150 read#2
chr1 101 150 read#3
chr1 101 150 read#4
chr1 105 154 read#5
chr1 200 249 read#6

$ cat reads.bed | genomic_overlaps subset -inv exon.bed
chr1 200 249 read#6
```
4.3 The genomic_scans tool

The genomic_scans tool (scan_reads in version 1.3) is used for window-based computations such as peak discovery (see Table 3). The command-line version offers several parameters for controlling the window size, statistical tests etc. Users with basic C/C++ skills can easily modify the source code to perform the statistical test of their choice using the GenomicRegionSetScanner class described in section 2.3 (page 1). For a usage example, see section 5.4 (page 61).

4.3.1 genomic_scans counts

| USAGE: | genomic_scans counts [OPTIONS] <REG-FILE> |
| DESCRIPTION: | Determines input read counts in sliding windows of reference regions. |
| DETAILS: | * Input formats: REG, GFF, BED, SAM  
| | * Operand: interval  
| | * Region requirements: single-interval  
| | * Region-set requirements: sorted by chromosome/strand/start |
| OPTIONS: | --help | help [true] |
| | -h | help [true] |
| | -v | verbose mode [false] |
| | -g | genome region file [genome.reg+] |
| | -r | reference region file [] |
| | -n | use genomic interval label as count [false] |
| | -min | minimum reads in window [10] |
| | -d | window distance [25] |
| | -w | window size (must be a multiple of window distance) [500] |

For an example, see section 5.3 (page 60).
4.3.2 genomic_scans peaks

```
USAGE:
genomic_scans peaks [OPTIONS] SIGNAL-REG-FILE CONTROL-REG-FILE [GENOME-UNIQ-REG-FILE]
DESCRIPTION:
Scans input reads to identify peaks.
DETAILS:
* Input formats: REG, GFF, BED, SAM
* Operand: interval
* Region requirements: single-interval
* Region-set requirements: sorted by chromosome/strand/start
OPTIONS:
--help help [true]
-h help [true]
-v verbose mode [false]
-g genome region file [genome.reg+]
-M method (binomial, poisson, binomial2) [binomial]
-n use genomic interval label as count [false]
-min minimum reads in window [10]
-d window distance [25]
-w window size (must be a multiple of window distance) [500]
-norm equalize background probabilities [false]
-cmp compare signal to control window [false]
-pval pvalue cutoff [1.000000]
-D print details [false]
```

Developers can easily modify the source code to perform the statistical test of their choice using the GenomicRegionSetScanner class described in 2.3 (page 9). For a usage example, see section 5.4 (page 61).
4.4 Permutation tests

This tool executes row permutations to determine p-values and q-values for all the categories contained in the input file (column #3, see input format below) given the statistic chosen by the user. More specifically, the statistic on the set of rows annotated by a given category is compared against the same statistic on permuted versions of the input on the value column (column #2). See also section 5.5. For details of the permutation test protocol see [7].

**USAGE:** permutation_test [OPTIONS] INPUT-FILE

**DESCRIPTION:** Execute permutation tests using various statistics.

**OPTIONS:**
- 
  -v     verbose mode             [false]
  -kmin  minimum support per category [10]
  -kmax  maximum support per category (default = no maximum) [0]
  -norm  normalize row values (if applicable) [false]
  -S     choose statistic [sum|n|sens|spec|ratio|t] [sum]
  -a     use a distribution for p-value approximation (not applicable to all statistics) [false]
  -u     find depleted categories (default = enriched) [false]
  -p     number of random permutations [100]
  -q     FDR cutoff [1.00]
  -f     print FDR instead of adjusted p-values [false]
  -h     print header [false]
  -d     print details [false]

**INPUT**

Every line in the input file has three tab-separated columns: label, value and a list of space-separated categories to be tested for significance:

```
$ head input.txt
Gene1  0.782  Tcell_development DNA_repair Notch_target
Gene2  0.102  metabolism
Gene3  0.231
```
### Supported statistics for a given category (-S option)

<table>
<thead>
<tr>
<th>Statistic</th>
<th>Description</th>
<th>Approximation (-a option)</th>
</tr>
</thead>
<tbody>
<tr>
<td>sum</td>
<td>the sum of values in a given category</td>
<td>not implemented</td>
</tr>
<tr>
<td>n</td>
<td>number of values &gt; 0</td>
<td>hypergeometric</td>
</tr>
<tr>
<td>sens</td>
<td>sensitivity</td>
<td>not implemented</td>
</tr>
<tr>
<td>spec</td>
<td>specificity</td>
<td>not implemented</td>
</tr>
<tr>
<td>ratio</td>
<td>mean of values divided by mean of values in the background</td>
<td>not implemented</td>
</tr>
<tr>
<td>t</td>
<td>t-test between mean of values against mean of values in the background</td>
<td>normal</td>
</tr>
</tbody>
</table>

### EXAMPLE

The output comprises one line per category sorted by q-value and p-value. Each line has 5 tab-separated fields: category, number of rows annotated by the category, q-value, p-value, and the value of the calculated statistic:

```
$ permutation_test -v -S n -a -p 10000 input.txt | head
```

<table>
<thead>
<tr>
<th>Category</th>
<th>Rows</th>
<th>q-value</th>
<th>p-value</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tcell_development</td>
<td>154</td>
<td>1.20e-04</td>
<td>3.4e-25</td>
<td>45.0000</td>
</tr>
<tr>
<td>Notch_target</td>
<td>102</td>
<td>5.40e-03</td>
<td>9.2e-15</td>
<td>20.0000</td>
</tr>
<tr>
<td>metabolism</td>
<td>4000</td>
<td>1.00e-00</td>
<td>2.7e-03</td>
<td>542.0000</td>
</tr>
</tbody>
</table>

For a more detailed example, see section 5.5 (page 62).
4.5 Vector operations

**USAGE:** vectors OPERATION [OPTIONS] <VECTORS>

**DESCRIPTION:** Perform vector operations using double-precision floating-point arithmetic.

**OPERATIONS:**

- **absmax** absolute maximum
- **-bins** create bins for vector values
- **-cutoff** apply cutoff
- **-div** divide
- **-exp** compute the exponential (inverse of log)
- **-fold** compute fold changes
- **-format** format vectors
- **-hist** histogram (all vectors)
- **-imax** maximum index
- **-imin** minimum index
- **-items** convert to itemset format
- **-log** compute the logarithm
- **-m** mean
- **-max** maximum
- **-med** median
- **-merge** merge consecutive vectors with identical labels
- **-min** minimum
- **-n** size
- **-norm** normalize
- **-pairs** create all vector pairs
- **-permute** permute order of vectors
- **-pow** compute the power
- **-q** quantiles
- **-rev** reverse
- **-sd** standard deviation
- **-shuffle** shuffle vector elements
- **-slide** create subvectors by a sliding window
- **-sort** sort
- **-sparse** convert to sparse format
- **-stat** statistics
- **-sum** sum
- **-test** test if greater than cutoff
### 4.6 Matrix operations

**USAGE:** matrix OPERATION [OPTIONS] <MATRIX>|MATRIX_FILE

**DESCRIPTION:** Perform matrix operations.

**OPERATIONS:**
- `-T` matrix transpose
- `-norm` normalization
- `-cnorm` column normalization
- `-rnorm` row normalization
- `-cstat` column statistics
- `-rstat` row statistics
- `-csum` column sums
- `-cutoff` apply cutoff
- `-test` test equality and/or inequality
- `-pairs` print all row pairs
- `-sparse` convert matrix to sparse format
- `-encode` print vectors as encoded sets
- `-shuffle` permute the elements of each row separately
- `-format` format matrix
- `-shrink` remove zero/empty columns
- `-mult` multiply matrix elements with coefficient
- `-rel` compute relative changes between element pairs in each row
- `-del` delete redundant rows
- `-sparse` convert to sparse format
- `-stat` statistics
- `-sum` sum
- `-test` test if greater than cutoff
5 Case study: a simple ChIP-seq pipeline

In this chapter, we demonstrate the utility of GenomicTools in constructing a simple pipeline for ChIP-seq analysis. The pipeline includes: (a) producing data for popular plots such as read profiles and read density heatmaps, (b) creating genome browser tracks for visualization, (c) identifying peaks as potential binding sites, and (d) performing an enrichment analysis. The following examples use UNIX command-line functions, but they also run on Cygwin under MS Windows.

5.1 Creating ChIP-seq read profiles

ChIP-seq read profiles are heavily used in ChIP-seq studies because they offer an easy method for data validation regarding the relative position of the ChIP-seq peaks (i.e. potential binding sites) with respect to chosen genomic features, such as gene transcriptional start sites (TSSs) or binding sites of other factors, such as enhancers. Additional validation is possible if expression data is available and the transcription factor or histone modification mark under ChIP-seq investigation is activating or repressive. In such a case, its read profile can be computed separately for genes of high vs. low expression and confirm its activating or repressive role.

Creating read profiles using GenomicTools is straightforward as demonstrated in the following example. First, the user creates the TSS regions using as input the gene transcript chromosomal coordinates in “genes.bed” which can be downloaded for example from the UCSC Genome Browser website. This is done using the genomic_regions tool “pos” and “shift” operations: the former chooses the 5’ end of gene transcripts (i.e. the TSS) and the latter performs a 10kb flanking operation upstream and downstream of the TSS:

```
$ head genes.bed
chr1 3044313 3044814 ENSMUSG00000090025:ENSMUST0000016944 1000 +
chr1 3092096 3092206 ENSMUSG00000064842:ENSMUST0000082908 1000 +
chr1 3546667 3503634 ENSMUSG00000089699:ENSMUST0000161581 1000 +
chr1 3670235 3671871 ENSMUSG00000073742:ENSMUST0000097833 1000 +
...
$ cat genes.bed | genomic_regions pos -op 5p | genomic_regions shiftp -5p -10000 -3p +10000 > TSS.10kb.bed
```

```
$ head TSS.10kb.bed
chr1 3034313 3054314 ENSMUSG00000090025:ENSMUST0000016944 1000 +
chr1 3082096 3102097 ENSMUSG00000064842:ENSMUST0000082908 1000 +
chr1 3446667 3466668 ENSMUSG00000089699:ENSMUST0000161581 1000 +
chr1 3660235 3680236 ENSMUSG00000073742:ENSMUST0000097833 1000 +
chr1 4509097 4529098 ENSMUSG00000064376:ENSMUST0000082442 1000 +
chr1 4787868 4807869 ENSMUSG00000025903:ENSMUST0000134384 1000 +
chr1 4787903 4807904 ENSMUSG00000025903:ENSMUST0000027036 1000 +
...
```
Next, the distances of the mapped ChIP-seq reads from the TSS regions are computed using the genomic_overlaps tool “offset” operation. The “offset” operation allows the user to choose a reference point for the query regions (“-op” option), and to express the computed offset as a fraction of the query region size (“-a” option) instead of an absolute number. Also, in this particular application, the strand information is ignored (“-i” option), because binding occurs both sense and anti-sense of the affected transcript.

\[ \text{head chipseq.bed} \]

```
chr1 3001228 3001229
chr1 3001228 3001229
chr1 3001438 3001439
```

\[ \text{cat chipseq.bed | genomic_overlaps offset -v -i -op 5p -a TSS.10kb.bed | cut -d' ' -f1 > offset.txt} \]

\[ \text{head offset.txt} \]

```
ENSMUSG00000090025:ENSMUST00000160944 0.007850
ENSMUSG00000090025:ENSMUST00000160944 0.007850
ENSMUSG00000090025:ENSMUST00000160944 0.021899
ENSMUSG00000090025:ENSMUST00000160944 0.021899
ENSMUSG00000090025:ENSMUST00000160944 0.030098
ENSMUSG00000090025:ENSMUST00000160944 0.030098
```

Finally, the computed offsets can be separated in genes of high vs. low expression, histogrammed using the vectors tool (operation “hist”) and plotted using R (see Figure 2 for a sample plot), Excel or any other similar tool or environment. For example, if the “offset.txt” file computed above was separated into two files “offset.high.txt” for the genes of high expression and “offset.low.txt” for the genes of low expression, then:

```
\[ \text{cat offset.high.txt | vectors -hist -n 6 -b 100 > profile.high.txt} \]
```

```
\[ \text{cat offset.low.txt | vectors -hist -n 6 -b 100 > profile.low.txt} \]
```

\[ \text{head offset.high.txt} \]

```
#bin-start  bin-freq    bin-counts
0.000000    0.008555    20676
0.010000    0.008522    20596
0.020000    0.008128    19644
```

Note that the histogram counts in column #3 need to be normalized by the number of genes in each group. Option “-n 6” sets the number of decimals to 6 and “-b 100” the number of histogram bins to 100.
5.2 Creating ChIP-seq read density heatmaps

Although average ChIP-seq profiles are useful for easy visualization and validation, they do not reveal the entire picture for every gene. This is achieved by ChIP-seq read density heatmaps around TSSs (Figure 3). To produce the data for this type of plot, the user can simply utilize the vectors operations “-merge” and “-bins”, so that now the histograms are produced per gene rather than for the entire offset file.

```bash
$ cat offset.high.txt | sort | vectors -merge | vectors -bins -b 200 -m 10 > heatmap.high.txt
$ head heatmap.high.txt
ENSMUSG00000090025:ENSMUST00000160944 0 0 0 0 0 0 0 0 0 0 0 0 ...
ENSMUSG00000064842:ENSMUST0000082908 0 0 0 0 0 0 0 0 0 0 0 0 ...
ENSMUSG0000051951:ENSMUST00000159265 0 0 0 0 0 0 0 4 0 4 0 4 ...
...
```

In the example above, we used a total of 200 bins (option “-b 200”), and a smoothing parameter “-m 10”, which sums the results in each series of consecutive 10 bins.
5.3 Creating window-based read densities

In ChIP-seq studies, researchers are interested in visualizing the densities of their ChIP-seq reads at a genome-wide scale so that they can understand the behavior of the studied protein in genes of interest. GenomicTools can be used to create window-based read densities to be displayed as a wiggle track in the UCSC Genome Browser (Figure 4). First, the user needs to decide on the window parameters: (a) the size of the window, (b) the distance between consecutive windows and (c) minimum number of read allowed in each window. The last two parameters establish a tradeoff between resolution and output file size. Here are some typical values:

```bash
$ set win_size = 500  # must be a multiple of win_dist
$ set win_dist = 25
$ set min_reads = 20
```

Then, the user needs to create a file describing the chromosomal bounds in REG or BED format:
The `genomic_scans` tool can be used to compute the counts of reads stored in “chipseq.bed” in sliding windows across the genome. Finally, the center of each window is computed and the “-wig” operation of `genomic_regions` converts to the wiggle format for display in the UCSC Genome Browser (Figure 4):

```
$ head chipseq.bed
chr1 3001228 3001229
chr1 3001228 3001229
chr1 3001438 3001439
...
$ cat chipseq.bed | genomic_scans counts -v -min $min_reads -w $win_size -d $win_dist -g genome.bed | genomic_regions center | genomic_regions wig -t "densities" -s $win_dist -c '0,0,150' > densities.wig
```

```
$ head densities.wig
track type=wiggle_0 name='densities' color=0,0,150
variableStep chrom=chr1 span=25
4775325 20
4775375 22
4775400 22
4775425 26
4775450 26
4775475 28
4775500 28
...
```

Figure 4: Example of window-based read densities in wiggle format.

### 5.4 Identifying window-based peaks

GenomicTools can also be used to identify window-based peaks and to display them as a BED track in the UCSC Genome Browser. This is achieved using the operation “peaks” of the `genomic_scans` tool. As in the “counts” operation in the example above, the user needs to determine the window size and distance as well as the minimum number of reads in the window. For each window, p-values are
computed using the binomial probability. A p-value cutoff can be enforced using the “-pval” option. The user can specify a file containing control reads (“control.bed” in our example below). If no control reads are specified, the computed p-values are based on a random background. Finally, the “bed” operation of the genomic_regions tool is used to convert the output to the BED format for visualization in the UCSC Genome browser (Figure 5, green track):

```
$ genomic_scans peaks -v -cmp -w $win_size -d $win_dist -min $min_reads -pval 1e-05 -g genome.bed chipseq.bed control.bed | genomic_regions -bed -t "peaks" -c '0,150,0' > peaks.bed
```

```
$ head peaks.bed
track name='peaks' itemRgb=On visibility=1
chr1 4775075 4775575 2.27e-09 10000 + 4775075 4775575 0,150,0
chr1 4775125 4775625 6.90e-11 10000 + 4775125 4775625 0,150,0
chr1 4775150 4775650 6.90e-11 10000 + 4775150 4775650 0,150,0
...
```

Figure 5: Example of window-based peaks in bed format.

### 5.5 Identifying enriched Gene Ontology terms

In a given biological context, for example a tissue type or disease state, certain proteins (transcription factors or modified histones) tend to bind to genes of specific functional categories. Gene enrichment analysis can identify these categories. In the GenomicTools platform, enrichment analysis is performed using the permutation_test tool. Using the ChIP-seq peaks computed above, we first calculate their densities across gene TSS regions - flanked by 10kb - using the “density” operation of the genomic_overlaps tool:

```
$ cat peaks.bed | genomic_overlaps density -v -i TSS.10kb.bed > tss.val
```

```
$ head tss.val
ENSMUSG00000090025:ENSMUST0000160944 0.0000e+00
ENSMUSG00000064842:ENSMUST0000082908 0.0000e+00
ENSMUSG00000051951:ENSMUST0000159265 0.0000e+00
...
```
Then, suppose we have a file containing gene annotations in a TAB-separated format where the first column is a gene id and the second column is a SPACE-separated list of annotations for the corresponding gene. The file must be sorted by gene id. For our example, we will use “gene.go” which contains annotations from the Gene Ontology [8]:

```
$ head gene.go
ENSMUSG00000000001      memb...rane_fusion ...
ENSMUSG00000000028      DNA-dependent DNA_replication ...
ENSMUSG00000000049      acylglycerol metabolism angiogenesis ...
ENSMUSG00000000058      M_phase_of_mitotic_cell_cycle ...
ENSMUSG00000000078      cytokine_and_chemokine_mediated_sign...
ENSMUSG00000000085      gene_silencing
ENSMUSG00000000093      aging cardiac_muscle_development ...
ENSMUSG00000000094      angiogenesis appendage_development ...
ENSMUSG00000000120      axon_guidance axonogenesis cell_proj...
ENSMUSG00000000125      Wnt_receptor_signaling_pathway ...
```

As an input, `permutation_test` needs to take a file containing 3 TAB-separated fields: the first field is an id (e.g. gene id), the second field is a value (e.g. density) and the third field is a SPACE-separated list of annotations. We first group the results in “tss.val” by gene using “vector-merge”, then choose the maximum density per gene across transcripts using “vector-max”, and finally perform a join operation with “gene.go” (note that the delimiter used in the join operation specified by option “-t” must be a TAB, i.e. Control-V-I):

```
$ cat tss.val | tr ':' ' ' | cut -f1,3 | sort | uniq | vectors-merge -n 6 | vectors-max -n 6 | join -a1 -t' ' -t gene.go > tss.val+go
```

```
$ head tss.val+go
ENSMUSG000000000544 0.000000
ENSMUSG000000000817 0.000000 I-kappaB_kinase/NF-kappaB_cascade ...
ENSMUSG000000001138 0.849960
ENSMUSG000000001143 1.549900
ENSMUSG000000001305 0.524970
ENSMUSG000000001674 0.399980
ENSMUSG000000002459 0.000000 negative_regulation_of_signal ...
ENSMUSG000000002881 0.774960 Schwann_cell_differentiation ...
ENSMUSG000000003134 0.049998 regulation_of_GTPase_activity ...
```

Now, we can run the `permutation_test` tool:

```
$ permutation_test -v -h -S n -a -p 10000 -q 0.05 tss.val+go > peaks.enriched.go.in.tss.txt
```

```
$ head peaks.enriched.go.in.tss.txt
CATEGORY CATEGORY-SIZE Q-VALUE P-VALUE STATISTIC
M_phase_of_mitotic_cell_cycle 13 0.00e+00 0.00e+00 13
regulation_of_lymphocyte_activation 12 6.92e-03 0.00e+00 12
```

63
<table>
<thead>
<tr>
<th>Pathway</th>
<th>Gene</th>
<th>Log10(p-value)</th>
<th>FDR</th>
<th>Stage</th>
</tr>
</thead>
<tbody>
<tr>
<td>translation</td>
<td>12</td>
<td>6.92e-03</td>
<td>0.00e+00</td>
<td>12</td>
</tr>
<tr>
<td>mitosis</td>
<td>13</td>
<td>6.92e-03</td>
<td>0.00e+00</td>
<td>13</td>
</tr>
<tr>
<td>membrane_lipid_metabolism</td>
<td>15</td>
<td>6.92e-03</td>
<td>1.38e-05</td>
<td>14</td>
</tr>
<tr>
<td>lymphocyte_activation</td>
<td>26</td>
<td>6.92e-03</td>
<td>1.45e-05</td>
<td>22</td>
</tr>
<tr>
<td>ubiquitin_cycle</td>
<td>13</td>
<td>2.27e-02</td>
<td>6.19e-05</td>
<td>12</td>
</tr>
<tr>
<td>protein_catabolism</td>
<td>13</td>
<td>2.27e-02</td>
<td>6.19e-05</td>
<td>12</td>
</tr>
<tr>
<td>T_cell_activation</td>
<td>20</td>
<td>2.27e-02</td>
<td>8.46e-05</td>
<td>17</td>
</tr>
</tbody>
</table>
References


