Genomic Regulation

Ivó Hernández, Helena Liz, Diego Fuentes, Fernando Freire

March 9, 2019

Contents

1 Genomic Regulation

1.1 Get chr16 CTCF segments

Get the *chr16* segments which share the same state between both monocyte replicates.

1.1.1 Test files

Script 1.1.1 (text)

```bash
1  %%bash
2  # Obtain files for test
3  cd files/tracks
4  cat Monocyte1_11_Master_11_segments.bed | grep 'chr16' | grep 'E9' | sort -k 2,3 -h | head
   ↪  -n 20 > monocyte1_segments.bed
5  cat Monocyte2_11_Master_11_segments.bed | grep 'chr16' | grep 'E9' | sort -k 2,3 -h | head
   ↪  -n 20 > monocyte2_segments.bed
6  echo "Monocyte 1 segments.bed"
7  cat monocyte1_segments.bed
8  wc -l monocyte1_segments.bed
9  echo "Monocyte 2 segments.bed"
10 cat monocyte2_segments.bed
11 wc -l monocyte2_segments.bed
```

Output

```
Monocyte 1 segments.bed
chr16        60400        61400        E9
chr16        72600        72800        E9
chr16        115200       116000       E9
chr16        146400       147400       E9
chr16        156600       157600       E9
chr16        167800       168200       E9
chr16        232200       232400       E9
chr16        412000       412600       E9
chr16        441800       442200       E9
chr16        486400       486800       E9
chr16        537600       538000       E9
chr16        597000       597600       E9
chr16        629000       629400       E9
chr16        661000       661600       E9
chr16        710800       711200       E9
chr16        711600       711800       E9
chr16        736200       736400       E9
chr16        761200       763200       E9
chr16        835400       836200       E9
chr16        1019400      1019600      E9
      20 monocyte1_segments.bed
Monocyte 2 segments.bed
chr16        60400        61400        E9
chr16        72400        72800        E9
chr16        115000       116400       E9
```

```
chr16          146600          147400          E9
chr16          155400          158200          E9
chr16          167800          168800          E9
chr16          231800          232200          E9
chr16          309000          309200          E9
chr16          353600          354200          E9
chr16          402200          403200          E9
chr16          412000          412800          E9
chr16          441800          442200          E9
chr16          508200          508400          E9
chr16          537400          538200          E9
chr16          596600          597400          E9
chr16          627800          630600          E9
chr16          660800          661800          E9
chr16          710800          711800          E9
chr16          717400          718200          E9
chr16          735800          736800          E9
        20 monocyte2_segments.bed
```

Script 1.1.2 (text)

```
1  %%bash
2  cd files/tracks
3  cat Monocyte1_11_Master_11_segments.bed | grep 'chr16' | grep 'E9' | sort -k 2,3 -h >
   ↪  monocyte1_segments.bed
4  cat Monocyte2_11_Master_11_segments.bed | grep 'chr16' | grep 'E9' | sort -k 2,3 -h >
   ↪  monocyte2_segments.bed
5  bedtools intersect -a monocyte1_segments.bed -b monocyte2_segments.bed > E9.bedtools.bed
6  head -n 20 E9.bedtools.bed
7
8  diff E9.bed E9.bedtools.bed
```

Output

```
chr16          60400          61400          E9
chr16          72600          72800          E9
chr16          115200          116000          E9
chr16          146600          147400          E9
chr16          156600          157600          E9
chr16          167800          168200          E9
chr16          412000          412600          E9
chr16          441800          442200          E9
chr16          537600          538000          E9
chr16          597000          597400          E9
chr16          629000          629400          E9
chr16          661000          661600          E9
chr16          710800          711200          E9
chr16          711600          711800          E9
chr16          736200          736400          E9
chr16          761200          763200          E9
chr16          835400          836200          E9
```

```
chr16          1105000         1105400          E9
chr16          1107200         1107800          E9
chr16          1286400         1287200          E9
```

Script 1.1.3 (text)

```
1  %%writefile files/test_tracks/bed1.bed
2  chr16       60400       61400       E9
3  chr16       72600       72800       E9
4  chr16       115200      116000      E9
5  chr16       146400      147400      E9
6  chr16       156600      157600      E9
7  chr16       167800      168200      E9
8  chr16       232200      232400      E9
9  chr16       412000      412600      E9
10 chr16       441800      442200      E9
11 chr16       486400      486800      E9
12 chr16       537600      538000      E9
13 chr16       597000      597600      E9
14 chr16       629000      629400      E9
15 chr16       661000      661600      E9
16 chr16       710800      711200      E9
17 chr16       711600      711800      E9
18 chr16       736200      736400      E9
19 chr16       761200      763200      E9
20 chr16       835400      836200      E9
21 chr16       1019400     1019600     E9
```

Output

```
Overwriting files/test_tracks/bed1.bed
```

Script 1.1.4 (text)

```
1  %%writefile files/test_tracks/bed2.bed
2  chr16       60400       61400       E9
3  chr16       72400       72800       E9
4  chr16       115000      116400      E9
5  chr16       146600      147400      E9
6  chr16       146610      147400      E8
7  chr16       155400      158200      E9
8  chr16       167800      168800      E9
9  chr16       231800      232200      E9
10 chr16       309000      309200      E9
11 chr16       353600      354200      E9
12 chr16       402200      403200      E9
13 chr16       412000      412800      E9
14 chr16       441800      442200      E9
15 chr16       508200      508400      E9
16 chr16       537400      538200      E9
```

```
17  chr16        596600        597400        E9
18  chr16        596700        597400        E8
19  chr16        596700        597400        E8
20  chr16        627800        630600        E9
21  chr16        660800        661800        E9
22  chr16        710800        711800        E9
23  chr16        717400        718200        E9
24  chr16        735800        736800        E9
```

Script 1.1.5 (text)

```
1  %%writefile files/test_tracks/dnase1.peaks.bed
2  chr1        770942        771278        chr1.9        584        .        0.039        1.79 ⌐
   ↪        -1        151
3  chr1        771678        771933        chr1.10        568        .        0.0343        1.5 ⌐
   ↪ 6        -1        121
4  chr1        773279        773398        chr1.11        555        .        0.0303        1.3 ⌐
   ↪ 8        -1        49
5  chr1        777497        777598        chr1.12        553        .        0.0299        1.3 ⌐
   ↪ 6        -1        46
6  chr1        794051        794336        chr1.13        569        .        0.0344        1.5 ⌐
   ↪ 7        -1        152
7  chr1        800514        800667        chr1.14        549        .        0.0287 ⌐
   ↪ 1.3        -1        34
8  chr1        805004        805656        chr1.15        1000        .        0.3561        16 ⌐
   ↪        -1        286
9  chr16        72620        73427        chr16.6        1000        .        0.2652        12. ⌐
   ↪ 5        -1        256
10 chr16        74047        74486        chr16.7        687        .        0.069 ⌐
   ↪ 3.2        -1        213
```

1.1.2 Methods

Script 1.1.6 (python)

```python
1  import re as re
2  import subprocess
3
4  def head(path, filename, lines=20):
5      """
```

5

```python
      """
      i = 0
      file = open(path + "/" + filename, "r")
      for line in file:
          print(line.strip())
          i += 1
          if i > lines:
              break
      file.close()

def get_parts(bed_line, sep='\t'):
      """
      """
      bed_line_parts = bed_line.rstrip('\n').split(sep)
      return bed_line_parts[0], int(bed_line_parts[1]), int(bed_line_parts[2]),
      ↪  bed_line_parts[3]

def concat_parts(chrom, start, end, feature, sep='\t'):
      """
      """
      bed_line = chrom + '\t' + str(start) +  '\t' + str(end) + '\t' + feature + '\n'
      return bed_line

def bed_coverage(path, filename, sep='\t'):
      """
      Returns the acummulated length of all the segments of the bed file filename
      """
      i = 0
      file = open(path + "/" + filename, "r")
      coverage = 0
      for line in file:
          _ , f1_segment_start, f1_segment_end, _ = get_parts(line)
          coverage += f1_segment_end + f1_segment_start
      file.close()
      return coverage

def bed_segment_count_by_re_feature(path, filename, re_feature, sep='\t'):
      """
      Returns the segment count by feature name of the bed file filename.
      The feature is informed as a regexp
      """
      i = 0
      file = open(path + "/" + filename, "r")
      segment_count = 0
      for line in file:
          _ , _, _, segment_feature = get_parts(line)
          if re.search(re_feature, segment_feature):
              segment_count += 1
      file.close()
      return segment_count
```

```python
def intersect_bed(input_dir, input_file1, input_file2, output_dir, output_file, chrom="chr16",
                  f1_feature_filter="E9", f2_feature_filter="E9", output_feature="E9",
                      sep='\t',
                  drop_feature_threshold=20, output_mode="intersect"):
    """
    If output node is intersect, returns the intersected bed segments
    If output mode is annotate, returns all the segments of input_file1
    annotated if it's the case with the feature defined in input_file2.
    """
    f1_segments = open(input_dir + "/" + input_file1, "r")
    f2_segments = open(input_dir + "/" + input_file2, "r")
    output_segments = open(output_dir + "/" + output_file, "w")
    f1_segment = f1_segments.readline()
    f2_segment = f2_segments.readline()
    while(f1_segment != "" and f2_segment != ""):
        f1_chrom, f1_segment_start, f1_segment_end, f1_feature = get_parts(f1_segment)
        f2_chrom, f2_segment_start, f2_segment_end, f2_feature = get_parts(f2_segment)
        # Filter f1 and read f1
        if f1_chrom != chrom or (f1_feature_filter != "" and f1_feature != f1_feature_filter):
            f1_segment = f1_segments.readline()
        # Filter f2 and read f2
        elif f2_chrom != chrom or (f2_feature_filter != "" and f2_feature !=
                f2_feature_filter):
            f2_segment = f2_segments.readline()
        # f2 segment downstream f1 segment
        elif f2_segment_start > f1_segment_end:
            if output_mode == "annotate" and drop_feature_threshold < f1_segment_end -
                    f1_segment_start:
                output_segment = concat_parts(chrom, f1_segment_start, f1_segment_end,
                        f1_feature)
                output_segments.write(output_segment)
            f1_segment = f1_segments.readline()
        # f1 segment downstream f2 segment
        elif f1_segment_start > f2_segment_end:
            f2_segment = f2_segments.readline()
        else: # Overlap
            # Save intersect
            if output_mode == "intersect":
                output_start = max(f1_segment_start, f2_segment_start)
                output_end =  min(f2_segment_end, f1_segment_end)
                if drop_feature_threshold < output_end - output_start:
                    output_segment = concat_parts(chrom, output_start, output_end,
                            output_feature)
                    output_segments.write(output_segment)
            # Advance f1
            if f2_segment_end >= f1_segment_end:
                f1_segment = f1_segments.readline()
            # Advance f2
            elif f1_segment_end > f2_segment_end:
                f2_segment = f2_segments.readline()
```

```
101              # Annotate mode: save f1, advance f1, advance f2
102          else:
103              if output_feature != "":
104                  feature = f1_feature + "+" + output_feature
105              else:
106                  feature = f1_feature
107              if drop_feature_threshold < f1_segment_end - f1_segment_start:
108                  output_segment = concat_parts(chrom, f1_segment_start, f1_segment_end,
     ↪   feature)
109                  output_segments.write(output_segment)
110              # Advance f1
111              f1_segment = f1_segments.readline()
112              # Advance f2
113              if f1_segment_end > f2_segment_end:
114                  f2_segment = f2_segments.readline()
115      while(output_mode == "annotate" and f1_segment != ""):
116          output_segments.write(f1_segment)
117          f1_segment = f1_segments.readline()
118
119      f1_segments.close()
120      f2_segments.close()
121      output_segments.close()
```

1.1.3 Tests

Script 1.1.7 (python)

```python
1  TEST_PATH = "files/test_tracks"
2  F1_FILE_TEST = "test1.bed"
3  F2_FILE_TEST = "test2.bed"
4  FILE_OUTPUT_TEST = "test_result.bed"
5  STATE = "E9"
6  CHROM = "chr16"
7  SEP = '\t'
8  DROP = 0
9
10 failed = 0
11 passed = 0
12 launched = 0
13
14 def create_testfile(segments, test_path, test_file):
15     """
16     """
17     output_segments = open(test_path + "/" + test_file, "w")
18     for segment in segments:
19         output_segment = concat_parts(segment[0], segment[1], segment[2], segment[3])
20         output_segments.write(output_segment)
21     output_segments.close()
22
23 def read_testfile(test_path, test_file):
24     """
```

```python
25          """
26          file_segments = open(test_path + "/" + test_file, "r")
27          segments = []
28          segment = file_segments.readline()
29          while(segment != ""):
30              chrom, segment_start, segment_end, feature = get_parts(segment)
31              segments.append([chrom, str(segment_start), str(segment_end), feature])
32              segment = file_segments.readline()
33          file_segments.close()
34          return segments
35
36  def do_test(test_number, segments_1, segments_2, test_expected_result,
37              verbose=True, mode="intersect", output_feature="output_feature",
38              test_path=TEST_PATH, f1_file_test=F1_FILE_TEST, f2_file_test=F2_FILE_TEST,
39              file_output_test=FILE_OUTPUT_TEST,
40              chrom=CHROM, f1_feature_filter="", f2_feature_filter="",
41              drop_feature_threshold=DROP, with_bedtools_check=False):
42      global failed, passed, launched
43      try:
44          launched += 1
45          create_testfile(segments_1, test_path, f1_file_test)
46          create_testfile(segments_2, test_path, f2_file_test)
47          intersect_bed(test_path, f1_file_test, f2_file_test, test_path,
48                        file_output_test, chrom=chrom,
49                        f1_feature_filter=f1_feature_filter,
                          ↪  f2_feature_filter=f2_feature_filter,
50                        output_feature=output_feature,
51                        sep=SEP, drop_feature_threshold=drop_feature_threshold,
                          ↪ output_mode=mode)
52          if verbose: head(test_path, file_output_test, 20)
53          output_segments = read_testfile(test_path, file_output_test)
54          if verbose: print("Threshold", drop_feature_threshold)
55          if verbose: print("Result", output_segments)
56          if verbose: print("Expected result", test_expected_result)
57          assert output_segments == test_expected_result, "Unexpected segments"
58          if with_bedtools_check:
59              # Check result agains bedtools
60              bedtools_cmd = "bedtools intersect"
61              if mode != "intersect" : bedtools_cmd += " -wa "
62              bedtools_cmd += " -a " + test_path + "/" + f1_file_test + " -b " + test_path +
                  ↪  "/" + f2_file_test + " > " + test_path + "/" + "bedtools_" + file_output_test
63              if verbose: print(bedtools_cmd)
64              p = subprocess.run(bedtools_cmd, shell = True, stdout = subprocess.PIPE)
65              if verbose: print("Bedtools", p.returncode, p.stdout)
66              assert p.returncode == 0 and p.stdout == b'', "Bedtools command has failed"
67              if mode != "intersect" :
68                  bedtools_cmd = "bedtools intersect -v"
69                  bedtools_cmd += " -a " + test_path + "/" + f1_file_test + " -b " + test_path
                      ↪  + "/" + f2_file_test + " >> " + test_path + "/" + "bedtools_" +
                      ↪  file_output_test
70                  if verbose: print(bedtools_cmd)
71                  p = subprocess.run(bedtools_cmd, shell = True, stdout = subprocess.PIPE)
```

```
72                    if verbose: print("Bedtools", p.returncode, p.stdout)
73                    assert p.returncode == 0 and p.stdout == b'', "Bedtools command 2 has failed"
74                    diff_cmd = "cat " + test_path + "/" + "bedtools_" + file_output_test + " |
          ↪    sort -k 1,3 -h | diff " + test_path + "/" + file_output_test + " -"
75                else:
76                    diff_cmd = "diff " + test_path + "/" + file_output_test + " " + test_path +
          ↪    "/" + "bedtools_" + file_output_test
77                if verbose: print(diff_cmd)
78                p = subprocess.run(diff_cmd, shell = True, stdout = subprocess.PIPE)
79                if verbose: print(diff_cmd)
80                if verbose: print("Diff", p.returncode, p.stdout)
81                assert p.returncode == 0 and p.stdout == b'', "Diff command has failed"
82            passed += 1
83            print ("Passed test %s" % (test_number))
84        except AssertionError:
85            print ("Failed test %s: Result:\n %s\nExpected result:\n %s\n"
86                    % (test_number, output_segments, test_expected_result))
87            failed += 1
88            exit(1)

89
90  # Test 1
91  segments_1 = [["chr16","0","100", "A"],
92                ["chr16","200","210", "A"]]
93  segments_2 = [["chr16","10","50", "B"]]
94  test_expected_result = [["chr16","10","50", "A"]]
95  do_test(1, segments_1, segments_2, test_expected_result, verbose = False,
    ↪   with_bedtools_check = True,
96          output_feature = "A")

97
98  # Test 1b
99  segments_1 = [["chr16","0","100", "A"],
100               ["chr16","200","210", "A"]]
101 segments_2 = [["chr16","10","50", "B"]]
102 test_expected_result = [["chr16","0","100", "A"],
103                         ["chr16","200","210", "A"]]
104 do_test(1, segments_1, segments_2, test_expected_result, verbose = True, with_bedtools_check
    ↪   = True,
105         output_feature = "", mode = "annotate")

106
107 # Test 2
108 test_expected_result = [["chr16","0","100", "A+output_feature"],
109                         ["chr16","200","210", "A"]]
110 do_test(2, segments_1, segments_2, test_expected_result, False, "annotate")

111
112 # Test 3
113 segments_1 = [["chr16","0","100", "A"],
114               ["chr16","200","210", "A"]]
115 segments_2 = [["chr16","10","20", "B"],
116               ["chr16","30","50", "B"]]
117 test_expected_result = [["chr16","0","100", "A+output_feature"],
118                         ["chr16","200","210", "A"]]
119 do_test(3, segments_1, segments_2, test_expected_result, False, "annotate")
```

10

```
120
121   # Test 4
122   segments_1 = [["chr16","0","100", "A"],
123               ["chr16","200","210", "A"]]
124   segments_2 = [["chr16","10","20", "B"],
125               ["chr16","30","50", "B"]]
126   test_expected_result = [["chr16","10","20", "output_feature"],
127                         ["chr16","30","50", "output_feature"]]
128   do_test(4, segments_1, segments_2, test_expected_result, False, "intersect")
129
130   # Test 5
131   segments_1 = [["chr16","0","100", "A"],
132               ["chr16","200","210", "A"]]
133   segments_2 = []
134   test_expected_result = []
135   do_test(5, segments_1, segments_2, test_expected_result, False, "intersect")
136
137   # Test 6
138   segments_1 = [["chr16","0","100", "A"],
139               ["chr16","200","210", "A"]]
140   segments_2 = []
141   test_expected_result = segments_1
142   do_test(6, segments_1, segments_2, test_expected_result, False, "annotate")
143
144   # Test 7
145   segments_1 = []
146   segments_2 = []
147   test_expected_result = segments_1
148   do_test(7, segments_1, segments_2, test_expected_result, False, "annotate")
149
150   # Test 8
151   segments_1 = []
152   segments_2 = []
153   test_expected_result = segments_1
154   do_test(8, segments_1, segments_2, test_expected_result, False, "intersect")
155
156   # Test 9
157   segments_1 = [["chr8","0","100", "A"],
158               ["chr8","100","150", "A"],
159               ["chr16","200","210", "A"],
160               ["chr16","300","1000", "A"]]
161   segments_2 = [["chr16","50","500", "B"],
162               ["chr16","600","800", "B"]]
163   test_expected_result = [["chr16","200","210", "A+output_feature"],
164                         ["chr16","300","1000", "A+output_feature"]]
165   do_test(9, segments_1, segments_2, test_expected_result, False, "annotate")
166
167   # Test 10
168   segments_1 = [["chr8","0","100", "A"],
169               ["chr8","100","150", "A"],
170               ["chr16","200","210", "A"],
171               ["chr16","300","1000", "A"]]
```

```python
172   segments_2 = [["chr16","50","500", "B"],
173                 ["chr16","600","800", "B"]]
174   test_expected_result = [["chr16","200","210", "output_feature"],
175                           ["chr16","300","500", "output_feature"],
176                           ["chr16","600","800", "output_feature"]]
177   do_test(10, segments_1, segments_2, test_expected_result, False, "intersect")
178
179   # Test 11
180   test_expected_result = [["chr16","300","500", "output_feature"],
181                           ["chr16","600","800", "output_feature"]]
182   do_test(11, segments_1, segments_2, test_expected_result, False, "intersect",
     ↪   drop_feature_threshold=30)
183
184   # Test 12
185   segments_1 = [["chr8","0","100", "A"],
186                 ["chr8","100","150", "A"],
187                 ["chr16","200","300", "A"],
188                 ["chr16","210","320", "A"]]
189   segments_2 = [["chr16","210","250", "B"],
190                 ["chr16","305","320", "B"]]
191   test_expected_result = [["chr16","210","250", "output_feature"],
192                           ["chr16","305","320", "output_feature"]]
193   do_test(12, segments_1, segments_2, test_expected_result, False, "intersect")
194
195   # Test 13
196   segments_1 = [["chr16","0","100", "E9"],
197                 ["chr16","200","300", "E9"]]
198   segments_2 = [["chr16","0","1000", "bbb"]]
199   test_expected_result = [["chr16","0","100", "E9+genes"],
200                           ["chr16","200","300", "E9+genes"]]
201   do_test(13, segments_1, segments_2, test_expected_result, True, "annotate",
     ↪   output_feature="genes")
202
203   print(" ")
204   if launched == passed: print("Passed All %s Test" %(passed))
205   else: print("ERROR: There are failed tests")
```

Output

```
Passed test 1
chr16          0          100          A
chr16          200        210          A
Threshold 0
Result [['chr16', '0', '100', 'A'], ['chr16', '200', '210', 'A']]
Expected result [['chr16', '0', '100', 'A'], ['chr16', '200', '210', 'A']]
bedtools intersect -wa  -a files/test_tracks/test1.bed -b files/test_tracks/test2.bed >
↪   files/test_tracks/bedtools_test_result.bed
Bedtools 0 b''
bedtools intersect -v -a files/test_tracks/test1.bed -b files/test_tracks/test2.bed >>
↪   files/test_tracks/bedtools_test_result.bed
Bedtools 0 b''
```

```
cat files/test_tracks/bedtools_test_result.bed | sort -k 1,3 -h | diff
↪    files/test_tracks/test_result.bed -
cat files/test_tracks/bedtools_test_result.bed | sort -k 1,3 -h | diff
↪    files/test_tracks/test_result.bed -
Diff 0 b''
Passed test 1
Passed test 2
Passed test 3
Passed test 4
Passed test 5
Passed test 6
Passed test 7
Passed test 8
Passed test 9
Passed test 10
Passed test 11
Passed test 12
chr16           0           100           E9+genes
chr16           200         300           E9+genes
Threshold 0
Result [['chr16', '0', '100', 'E9+genes'], ['chr16', '200', '300', 'E9+genes']]
Expected result [['chr16', '0', '100', 'E9+genes'], ['chr16', '200', '300', 'E9+genes']]
Passed test 13

Passed All 14 Test
```

1.1.4 CTCF segments

Script 1.1.8 (python)

```python
PATH = "files/tracks"
M1_FILE = "Monocyte1_11_Master_11_segments.bed"
M2_FILE = "Monocyte2_11_Master_11_segments.bed"
CHROM = "chr16"
STATE = "E9"
intersect_bed(PATH, M1_FILE, M2_FILE, PATH, STATE + ".bed", chrom=CHROM,
                  f1_feature_filter = STATE, f2_feature_filter = STATE, output_feature =
                  ↪   STATE, sep=SEP,
                  drop_feature_threshold = 10)

head(PATH, STATE + ".bed", 10)
print("Output file:", STATE + ".bed")
```

Output

```
chr16        60400         61400        E9
chr16        72600         72800        E9
chr16        115200        116000       E9
chr16        146600        147400       E9
chr16        156600        157600       E9
```

13

```
chr16          167800          168200          E9
chr16          412000          412600          E9
chr16          441800          442200          E9
chr16          537600          538000          E9
chr16          597000          597400          E9
chr16          629000          629400          E9
Output file: E9.bed
```

1.1.5   Test against bedtools

We check there is no difference between the intersect file generated by bedtools and our file

Script 1.1.9 (text)

```bash
%%bash
cd files/tracks
cat Monocyte1_11_Master_11_segments.bed | grep 'chr16' | grep 'E9' | sort -k 2,3 -h >
↪  monocyte1_segments.bed
cat Monocyte2_11_Master_11_segments.bed | grep 'chr16' | grep 'E9' | sort -k 2,3 -h >
↪  monocyte2_segments.bed
bedtools intersect -a monocyte1_segments.bed -b monocyte2_segments.bed > E9.bedtools.bed
```

Script 1.1.10 (python)

```python
p = subprocess.run("diff files/tracks/E9.bed files/tracks/E9.bedtools.bed", shell=True,
↪  stdout=subprocess.PIPE)
#print(p.returncode, p.stdout)
assert p.returncode == 0 and p.stdout == b'', "The files are different"
```

1.2   Segment annotation

Annotate the segments. At a minimum, the percentage of segments that overlap with protein-coding genes in said chromosome should be given.

1.2.1   Tracks to annotate

The tracks are obtained from https://genome.ucsc.edu/cgi-bin/hgTables

1.2.2   Annotate gene overlap

Script 1.2.1 (python)

```python
OUTPUT_FEATURE = "genes"
INPUT_FILE = STATE + ".bed"
OUTPUT_FILE = STATE + "_" + OUTPUT_FEATURE + ".bed"
ANNOTATION_TRACK = "hg19_genes_sorted.bed"
intersect_bed(PATH, INPUT_FILE, ANNOTATION_TRACK,
              PATH, OUTPUT_FILE, chrom = CHROM,
```

Figure 1: Table browser

```
7                    f1_feature_filter = "", f2_feature_filter="",
8                    output_feature = OUTPUT_FEATURE, sep=SEP,
9                    drop_feature_threshold = 0, output_mode="annotate")
10  head(PATH, OUTPUT_FILE, 10)
11
12  overlap_segment_count = bed_segment_count_by_re_feature(PATH, OUTPUT_FILE, OUTPUT_FEATURE)
13  print("")
14  print("Count of state segments overlapped:", overlap_segment_count)
15  total_segment_count = bed_segment_count_by_re_feature(PATH, OUTPUT_FILE, "")
16  print("Count of all state segments", total_segment_count)
17  print("Percent overlapped state segments over total segments:",
18          overlap_segment_count * 100 / total_segment_count)
19  print("Output file:", OUTPUT_FILE)
```

Output

```
chr16         60400          61400          E9
chr16         72600          72800          E9
chr16         115200          116000          E9+genes
chr16         146600          147400          E9+genes
chr16         156600          157600          E9+genes
chr16         167800          168200          E9+genes
chr16         412000          412600          E9
chr16         441800          442200          E9+genes
chr16         537600          538000          E9+genes
chr16         597000          597400          E9+genes
chr16         629000          629400          E9+genes


Count of state segments overlapped: 220
Count of all state segments 468
```

```
Percent overlapped state segments over total segments: 47.00854700854701
Output file: E9_genes.bed
```

Test against bedtools We check there is no difference between the intersect file generated by bedtools and our file

```bash
%%bash
cd files/tracks
bedtools annotate -i E9.bed -files hg19_genes_sorted.bed | sort -k1,1 -k2,3n | grep -v
  ↪  '0.000000' | cut -f 1,2,3 > E9_genes.bedtools.bed
grep 'genes' E9_genes.bed | cut -f 1,2,3 | diff - E9_genes.bedtools.bed
error=$?
if [ $error -eq 2 ]
then
    echo "Error in diff command"
elif [ $error -eq 1 ]
then
    echo "Files differ"
else
    echo "OK"
fi
```

Output

```
OK
```

1.2.3   Annotate exon overlap

Script 1.2.3 (python)

```python
OUTPUT_FEATURE = "exons"
INPUT_FILE = STATE + "_gene.bed"
OUTPUT_FILE = STATE + "_" + OUTPUT_FEATURE + "_gene.bed"
ANNOTATION_TRACK = "hg19_coding_exons_sorted.bed"
intersect_bed(PATH, INPUT_FILE, ANNOTATION_TRACK,
              PATH, OUTPUT_FILE, chrom = CHROM,
              f1_feature_filter = "", f2_feature_filter="",
              output_feature = OUTPUT_FEATURE, sep=SEP,
              drop_feature_threshold = 10, output_mode="annotate")
head(PATH, OUTPUT_FILE, 10)

overlap_segment_count = bed_segment_count_by_re_feature(PATH, OUTPUT_FILE, OUTPUT_FEATURE)
print("")
print("Count of state segments overlapped:", overlap_segment_count)
total_segment_count = bed_segment_count_by_re_feature(PATH, OUTPUT_FILE, "")
print("Count of all state segments", total_segment_count)
print("Percent overlapped state segments over total segments:",
```

```
18        overlap_segment_count * 100 / total_segment_count)
19  print("Output file:", OUTPUT_FILE)
```

Output

chr16   60400   61400   E9
chr16   72600   72800   E9
chr16   115200   116000   E9+gene
chr16   146600   147400   E9+gene
chr16   156600   157600   E9+gene
chr16   167800   168200   E9+gene
chr16   412000   412600   E9
chr16   441800   442200   E9+gene
chr16   537600   538000   E9+gene
chr16   597000   597400   E9+gene+exons
chr16   629000   629400   E9+gene+exons

Count of state segments overlapped: 41
Count of all state segments 468
Percent overlapped state segments over total segments: 8.760683760683762
Output file: E9_exons_gene.bed

1.2.4 Annotate upstream 200 overlap

Script 1.2.4 (python)

```
1   OUTPUT_FEATURE = "up200"
2   INPUT_FILE = STATE + "_gene.bed"
3   OUTPUT_FILE = STATE + "_" + OUTPUT_FEATURE + "_exons_gene.bed"
4   ANNOTATION_TRACK = "hg19_up200_sorted.bed"
5   intersect_bed(PATH, INPUT_FILE, ANNOTATION_TRACK,
6                 PATH, OUTPUT_FILE, chrom = CHROM,
7                 f1_feature_filter = "", f2_feature_filter="",
8                 output_feature = OUTPUT_FEATURE, sep=SEP,
9                 drop_feature_threshold = 10, output_mode="annotate")
10  head(PATH, OUTPUT_FILE, 10)
11
12  overlap_segment_count = bed_segment_count_by_re_feature(PATH, OUTPUT_FILE, OUTPUT_FEATURE)
13  print("")
14  print("Count of state segments overlapped:", overlap_segment_count)
15  total_segment_count = bed_segment_count_by_re_feature(PATH, OUTPUT_FILE, "")
16  print("Count of all state segments", total_segment_count)
17  print("Percent overlapped state segments over total segments:",
18        overlap_segment_count * 100 / total_segment_count)
19  print("Output file:", OUTPUT_FILE)
```

Output

```
chr16          60400          61400          E9+up200
chr16          72600          72800          E9+up200
chr16          115200         116000            E9+gene
chr16          146600         147400            E9+gene
chr16          156600         157600            E9+gene
chr16          167800         168200            E9+gene
chr16          412000         412600          E9
chr16          441800         442200            E9+gene
chr16          537600         538000            E9+gene
chr16          597000         597400            E9+gene
chr16          629000         629400            E9+gene


Count of state segments overlapped: 20
Count of all state segments 468
Percent overlapped state segments over total segments: 4.273504273504273
Output file: E9_up200_exons_gene.bed
```

1.3   DNASE I overlap

Download the peaks of DNase I in monocytes of ENCODE for chr16 and calculate the percentage of overlap between DNaseI-peaks and your work segments. Use the file wgEncodeOpenChromDnaseMonocd14Pk.narrowPeak.gz in: http://hgdownload.cse.ucsc.edu/goldenpath/hg19/encodeDCC/wgEncodeOpenChromDnase

1.3.1   Tests

Script 1.3.1 (python)

```python
intersect_bed(TEST_PATH, "bed1.bed", "dnase1.peaks.bed", TEST_PATH,
                STATE + "_dnase1_test.bed",
                chrom = CHROM, f1_feature_filter = STATE, f2_feature_filter = "",
                output_feature = "dnase1", sep=SEP,
                drop_feature_threshold=0, output_mode="")
head(TEST_PATH, STATE +"_dnase1_test.bed", 10)
```

Output

```
chr16          72600          72800          E9+dnase1
```

1.3.2   Overlap

Overlap by coverage   Overlap calculated as percent ratio between sum of base pair overlapped and sum of total base pair covered by all the E9 segments. This method doesn't have much sense because it depends of the arbitrary sensitivity of the dna base segments (200 bps in the case of chromatin states).

Output

```
chr16        72620        72800        E9_dnase1_overlap
chr16        115448       116000       E9_dnase1_overlap
chr16        146819       147400       E9_dnase1_overlap
chr16        157056       157367       E9_dnase1_overlap
chr16        167800       168118       E9_dnase1_overlap
chr16        412000       412600       E9_dnase1_overlap
chr16        441800       442200       E9_dnase1_overlap
chr16        537761       538000       E9_dnase1_overlap
chr16        597000       597400       E9_dnase1_overlap
chr16        629000       629400       E9_dnase1_overlap
chr16        661000       661548       E9_dnase1_overlap


Coverage DNASE peaks: 22653113851288 bps
Coverage E9: 43366167200 bps
Percent overlap over total coverage peaks: 0.19143578884866774
Output file: E9_dnase1.bed
```

Overlap by segment count   Overlap calculated as percent ratio between segment count of overlapped E9-DNASE segments and total count of E9 segments.

Script 1.3.3 (python)

```python
1  OUTPUT_FEATURE = "dnase1"
2
3  intersect_bed(PATH, STATE + ".bed", "wgEncodeOpenChromDnaseMonocd14Pk.narrowPeak.bed",
4                PATH, STATE + "_dnase1.bed", chrom = CHROM,
5                f1_feature_filter = STATE, f2_feature_filter="",
6                output_feature = OUTPUT_FEATURE, sep=SEP,
7                drop_feature_threshold = 10, output_mode="annotate")
8  head(PATH, STATE + "_dnase1.bed", 10)
9
10 overlap_segment_count = bed_segment_count_by_re_feature(PATH, STATE + "_dnase1.bed",
```

19
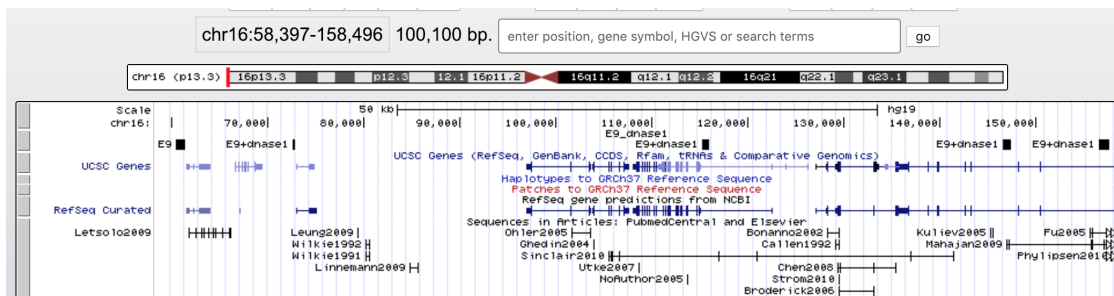
Figure 2

```
11                                      OUTPUT_FEATURE)
12  print("")
13  print("Count of state segments overlapped:", overlap_segment_count)
14  total_segment_count = bed_segment_count_by_re_feature(PATH, STATE + "_dnase1.bed", "")
15  print("Count of all state segments", total_segment_count)
16  print("Percent overlapped state segments over total segments:",
17          overlap_segment_count * 100 / total_segment_count)
18  print("Output file:", STATE + "_dnase1.bed")
```

Output

```
chr16       60400       61400       E9
chr16       72600       72800       E9+dnase1
chr16       115200      116000      E9+dnase1
chr16       146600      147400      E9+dnase1
chr16       156600      157600      E9+dnase1
chr16       167800      168200      E9+dnase1
chr16       412000      412600      E9+dnase1
chr16       441800      442200      E9+dnase1
chr16       537600      538000      E9+dnase1
chr16       597000      597400      E9+dnase1
chr16       629000      629400      E9+dnase1


Count of state segments overlapped: 342
Count of all state segments 468
Percent overlapped state segments over total segments: 73.07692307692308
Output file: E9_dnase1.bed
```

Automated verifications

Script 1.3.4 (python)

```
1  segment_count_annotate = bed_segment_count_by_re_feature(PATH, STATE + "_dnase1.bed", "")
2  segment_count = bed_segment_count_by_re_feature(PATH, STATE + ".bed", "")
3  assert segment_count_annotate == segment_count,\
4          "Count of annotated segments not equal to count of original segments"
```

20

Visual inspection

```
1  %%bash
2  export TRACKS=files/tracks/
3  echo "Counts"
4  wc -l ${TRACKS}E9_dnase1.bed
5  wc -l ${TRACKS}E9.bed
6  tail ${TRACKS}E9_dnase1.bed
7  echo
8  tail ${TRACKS}E9.bed
9  echo
10 head ${TRACKS}E9_dnase1.bed
11 echo
12 head ${TRACKS}E9.bed
13 echo
14 echo "Counts of annotations:"
15 cat ${TRACKS}E9_dnase1.bed | grep "dnase1" | wc -l
16 cat ${TRACKS}E9.bed | grep "" | wc -l
```

Output

```
Counts
     468 files/tracks/E9_dnase1.bed
     468 files/tracks/E9.bed
chr16        89233600        89234800        E9+dnase1
chr16        89527000        89527400        E9
chr16        89623800        89624200        E9+dnase1
chr16        89707800        89708000        E9+dnase1
chr16        89772400        89772600        E9+dnase1
chr16        89927000        89927800        E9+dnase1
chr16        89976600        89977000        E9+dnase1
chr16        90092400        90092800        E9+dnase1
chr16        90182400        90183000        E9
chr16        90281600        90282000        E9

chr16        89233600        89234800        E9
chr16        89527000        89527400        E9
chr16        89623800        89624200        E9
chr16        89707800        89708000        E9
chr16        89772400        89772600        E9
chr16        89927000        89927800        E9
chr16        89976600        89977000        E9
chr16        90092400        90092800        E9
chr16        90182400        90183000        E9
chr16        90281600        90282000        E9

chr16        60400        61400        E9
chr16        72600        72800        E9+dnase1
chr16        115200        116000        E9+dnase1
chr16        146600        147400        E9+dnase1
chr16        156600        157600        E9+dnase1
```

```
    chr16           167800          168200          E9+dnase1
    chr16           412000          412600          E9+dnase1
    chr16           441800          442200          E9+dnase1
    chr16           537600          538000          E9+dnase1
    chr16           597000          597400          E9+dnase1

    chr16           60400           61400           E9
    chr16           72600           72800           E9
    chr16           115200          116000          E9
    chr16           146600          147400          E9
    chr16           156600          157600          E9
    chr16           167800          168200          E9
    chr16           412000          412600          E9
    chr16           441800          442200          E9
    chr16           537600          538000          E9
    chr16           597000          597400          E9

    Counts of annotations:
        342
        468
```

Analysis

Biological background   *Insulation subsystem*
Insulator genomic function refers either to a barrier function or an enhancer-blocking function:

1. Barrier function. In cellular division heterochromatin and euchromatin must to be insulated from each other to prevent undesirable gene expression, for instance to prevent the inactive heterochromatic domain from erroneously inhibiting genes in the active euchromatic domain and vice versa.

2. Enhancer-blockig function. The activity of a given gene is controlled by enhancer sequences, which can be found either adjacent to the gene promoter or at a considerable distance either downstream or upstream of the gene. Distance at the level of the linear genome does not impose a problem for enhancer function as the intervening DNA is looped out such that promoter and enhancer will be in close contact. Insulators play a role in the three-dimensional folding of chromatin, allowing for or preventing functional contact between enhancer and promoter elements.

DNASE I
Deoxyribonuclease I (DNase I), is an endonuclease coded by the human gene DNASE1.
DNase I is a nuclease that cleaves DNA preferentially at phosphodiester linkages adjacent to a pyrimidine nucleotide. It seems to play a role in DNA fragmentation during apoptosis.
DNase I hypersensitive sites are thought to be characterized by open, accessible chromatin, so, a DNase I peak enable the identification of regions of the genome which are likely to contain active genes.
CTCF
11-zinc finger protein or nuclear protein CCCTC-binding factor (CTCF) is a transcription factor encoded by the CTCF gene in humans.
When bound to insulator sequences can prevent undesirable crosstalk between active and inactive genomic regions, and it can also shield particular genes from enhancer function, a role that has many appli-

10 µm

Multipotential hematopoietic stem cell (Hemocytoblast)

Common myeloid progenitor

Common lymphoid progenitor

Bone marrow

Megakaryoblast

Proerythroblast (Pronormoblast)

Myeloblast

Lymphoblast

Promegakaryocyte

Basophilic erythroblast

B. promyelocyte

N. promyelocyte

E. promyelocyte

Monoblast

Polychromatic erythroblast

B. myelocyte

N. myelocyte

E. myelocyte

Promonocyte

Prolymphocyte

Orthochromatic erythroblast (Normoblast)

B. metamyelocyte

N. metamyelocyte

E. metamyelocyte

Megakaryocyte

Polychromatic erythrocyte [1] (Reticulocyte)

B. band

N. band

E. band

Blood

Thrombocytes (Platelets)

Erythrocyte [2] (Red blood cell)

Basophil

Neutrophil

Eosinophil

Monocyte

Small lymphocyte [4]

Natural killer cell (Large granular lymphocyte)

Thrombopoiesis

Erythropoiesis

Granulopoiesis

Monocytopoiesis

Tissue

Mast cell

Macrophage

Myeloid dendritic cell [3]

B lymphocyte

T lymphocyte

Lymphopoiesis

Plasma cell

Lymphoid dendritic cell [3]
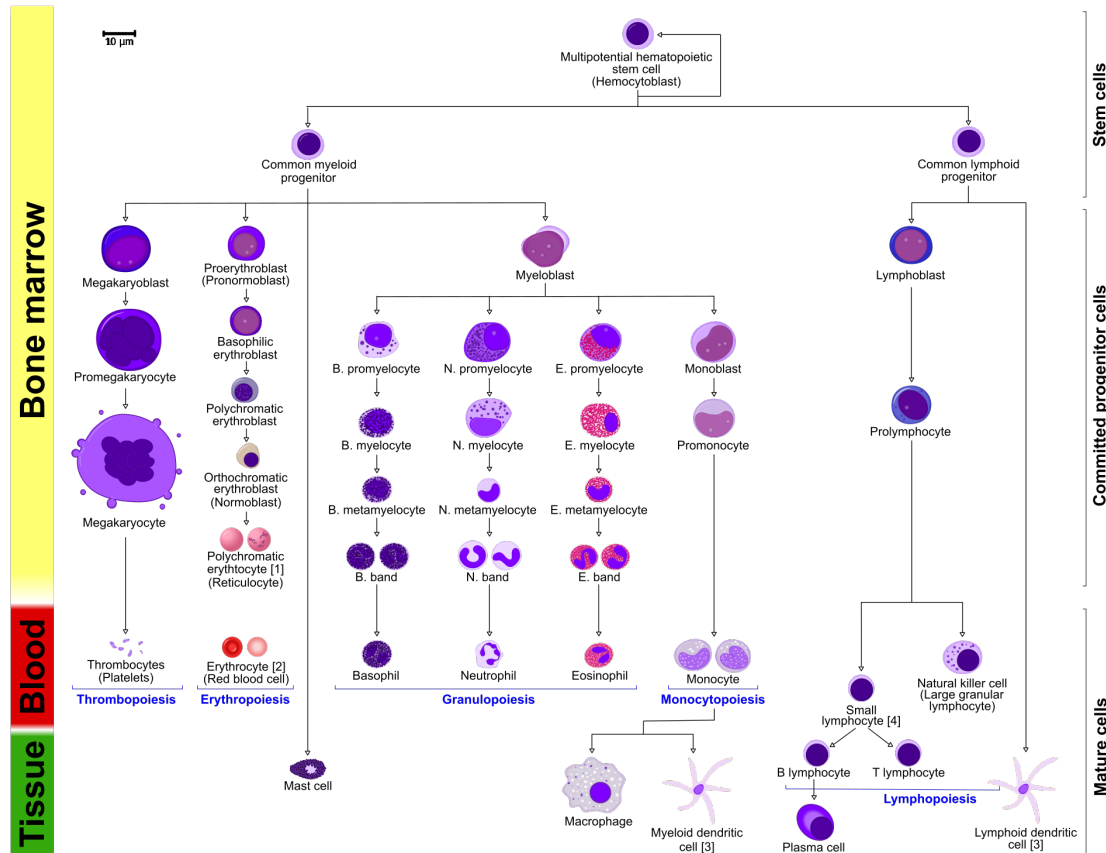
Stem cells

Committed progenitor cells

Mature cells

Figure 3: Human hematopoiesis

cations in development. Exciting recent work has demonstrated roles for CTCF in, for example, embryonic, neuronal and haematopoietic development.

Its main roles are:

1. Insulation by binding of targeting insulator sequence elements located between enhancer and promoter sequences, blocking the interaction of transcription factors.

2. Insulation by altering the 3D structure of chromatin, forming open chromatin loops, remodeling the heterochromatin structures and therefore preventing repressive heterochromatin actuation into a neighbouring domain.

Hematopoiesis   This diagram shows the hematopoiesis as it occurs in humans. The morphological characteristics of the hematopoietic cells are shown as seen in a Wright's stain, May-Giemsa stain or May-Grünwald-Giemsa stain. Alternative names of certain cells are indicated between parentheses. Certain cells may have more than one characteristic appearance. In these cases, more than one representation of the same cell has been included. Together, the monocyte and the lymphocytes comprise the agranulocytes, as opposed to the granulocytes (basophil, neutrophil and eosinophil) that are produced during granulopoiesis. B., N. and E. stand for Basophilic, Neutrophilic and Eosinophilic, respectively – as in Basophilic promyelocyte. For lymphocytes, the T and B are actual designations.

The polychromatic erythrocyte (reticulocyte) at the right shows its characteristic appearance when stained with methylene blue or Azure B.

Figure 4: E9 vs DNaseI blood cell types

The erythrocyte at the right is a more accurate representation of its appearance in reality when viewed through a microscope.

Other cells that arise from the monocyte: osteoclast, microglia (central nervous system), Langerhans cell (epidermis), Kupffer cell (liver).

The T and B lymphocyte are split to better indicate that the plasma cell arises from the B-cell. Note that there is no difference in the appearance of B- and T-cells unless specific staining is applied.

Analysis of data   We found that 70% of the E9 segments also contain DNAse I anchoring sequences, which requires regions of open chromatin. Can this indicate that in these areas CTCF is performing the second of the functions as an insulator: the enhacer-blocking factor?

1.4   Display in genome browser

Visualize (and show) a region of the genome in the UCSC browser where at least one of your segments can be seen (upload the track generated by ChromHMM) and DNaseI in all cell types of blood ENCODE.

First attempt: with the tracks available in USCS genome browser:

Second attempt:

1. Download from ENCODE https://www.encodeproject.org/metadata/type=Experiment&assay_title=DNase-seq&replicates.library.biosample.donor.organism.scientific_name=Homo+sapiens&files.file_type=bed+narrowPe

2. Open metadata.tsv in excell and search for a blood cell experiment (for instance, 808 row) and download the track https://www.encodeproject.org/files/ENCFF304TBE/@@download/ENCFF304TBE.bed.gz

3. Pending: upload the track and download more tracks.

1.5   Search of motifs.

1.5.1   Strategy

Segments in state 9 must contain CTCF binding sequences and probably enhancer, DNase 1 and other TF binding sequences.

It would be interesting to find at least the consensus sequence of CTCF binding sites.

It was done in this article:

https://www.ncbi.nlm.nih.gov/pmc/articles/PMC2572726/#!po=17.5676

But it's a *de novo* discovery.

Another approach (or another interpretation of the question and more easy) is to search in the segments which of them contains a motif corresponding to the PSM of CTCF binding sequence. It could be a good strategy to validate the quality of the E9 segments.

1.6   Overlapping with methylation regions

Calculate the% overlap with hyper- (Methylation> 0.75) or hypo-methylated (Methylation <0.25) regions in available monocytes in the BLUEPRINT portal DCC. (http://dcc.blueprint-epigenome.eu/#/home) BED files belonging to the donor C001UY.

1.6.1 Procedure

1.6.2 Obtain the bed segments.

...bla bla bla

1.6.3 Annotation

We have made the annotation with R-packages. Alternatively we check the results with our own software. We calculate here the

Script 1.6.1 (python)

```python
OUTPUT_FEATURE = "hypo"

intersect_bed(PATH, STATE + ".bed", "hypo.bed",
                PATH, STATE + "_hypo.bed", chrom = CHROM,
                f1_feature_filter = STATE, f2_feature_filter="",
                output_feature = OUTPUT_FEATURE, sep=SEP,
                drop_feature_threshold = 0, output_mode="annotate")
head(PATH, STATE + "_hypo.bed", 10)

overlap_segment_count = bed_segment_count_by_re_feature(PATH, STATE + "_hypo.bed",
                                                OUTPUT_FEATURE)
print("")
print("Count of state segments overlapped hypo:", overlap_segment_count)
total_segment_count = bed_segment_count_by_re_feature(PATH, STATE + "_hypo.bed", "")
print("Count of all state segments", total_segment_count)
print("Percent overlapped state segments over total segments:",
        overlap_segment_count * 100 / total_segment_count)
print("Output file:", STATE + "_hypo.bed")
```

Output

```
chr16        60400        61400        E9+hypo
chr16        72600        72800        E9
chr16        115200        116000        E9+hypo
chr16        146600        147400        E9+hypo
chr16        156600        157600        E9
chr16        167800        168200        E9+hypo
chr16        412000        412600        E9+hypo
chr16        441800        442200        E9
chr16        537600        538000        E9
chr16        597000        597400        E9
chr16        629000        629400        E9+hypo

Count of state segments overlapped hypo: 288
Count of all state segments 468
Percent overlapped state segments over total segments: 61.53846153846154
Output file: E9_hypo.bed
```

Script 1.6.2 (python)

```python
OUTPUT_FEATURE = "hyper"

intersect_bed(PATH, STATE + "_hypo.bed", "hyper.bed",
                  PATH, STATE + "_hypo_hyper.bed", chrom = CHROM,
                  f1_feature_filter = "", f2_feature_filter="",
                  output_feature = OUTPUT_FEATURE, sep=SEP,
                  drop_feature_threshold = 0, output_mode="annotate")
head(PATH, STATE + "_hypo_hyper.bed", 10)

overlap_segment_count = bed_segment_count_by_re_feature(PATH, STATE + "_hypo_hyper.bed",
                                                        OUTPUT_FEATURE)
print("")
print("Count of state segments overlapped hyper:", overlap_segment_count)
total_segment_count = bed_segment_count_by_re_feature(PATH, STATE + "_hypo_hyper.bed", "")
print("Count of all segments", total_segment_count)
print("Percent overlapped hyper segments over total segments:",
      overlap_segment_count * 100 / total_segment_count)
print("Output file:", STATE + "_hypo_hyper.bed")
```

Output

```
chr16          60400          61400          E9+hypo+hyper
chr16          72600          72800          E9+hyper
chr16          115200         116000         E9+hypo+hyper
chr16          146600         147400         E9+hypo+hyper
chr16          156600         157600         E9+hyper
chr16          167800         168200         E9+hypo+hyper
chr16          412000         412600         E9+hypo+hyper
chr16          441800         442200         E9+hyper
chr16          537600         538000         E9+hyper
chr16          597000         597400         E9
chr16          629000         629400         E9+hypo


Count of state segments overlapped hyper: 322
Count of all segments 468
Percent overlapped hyper segments over total segments: 68.80341880341881
Output file: E9_hypo_hyper.bed
```

Script 1.6.3 (python)

```python
overlap_segment_count = bed_segment_count_by_re_feature(PATH, STATE + "_hypo_hyper.bed",
                                                        r"hypo\+hyper")
print("")
print("Count of state segments overlapped hypo+hyper:", overlap_segment_count)
total_segment_count = bed_segment_count_by_re_feature(PATH, STATE + "_hypo_hyper.bed", "")
print("Count of all segments", total_segment_count)
print("Percent overlapped hypo+hyper segments over total segments:",
      overlap_segment_count * 100 / total_segment_count)
```

Output

```
Count of state segments overlapped hypo+hyper: 192
Count of all segments 468
Percent overlapped hypo+hyper segments over total segments: 41.02564102564103
```

1.6.4 Verification against bedtools annotation

Script 1.6.4 (text)

```
1  %%bash
2  cd files/tracks
3  bedtools annotate -i E9.bed -files hypo.bed | sort -k1,1 -k2,3n | grep -v '0.000000' | cut
   ↪  -f 1,2,3 > E9_hypo.bedtools.bed
4  grep 'hypo' E9_hypo_hyper.bed | cut -f 1,2,3 | diff - E9_hypo.bedtools.bed > /dev/null 2>&1
5  error=$?
6  if [ $error -eq 2 ]
7  then
8      echo "Error in diff command"
9  elif [ $error -eq 1 ]
10 then
11     echo "Files differ"
12 else
13     echo "OK"
14 fi
```

Output

```
OK
```

Script 1.6.5 (text)

```
1  %%bash
2  cd files/tracks
3  bedtools annotate -i E9.bed -files hyper.bed | sort -k1,1 -k2,3n | grep -v '0.000000' | cut
   ↪  -f 1,2,3 > E9_hyper.bedtools.bed
4  grep 'hyper' E9_hypo_hyper.bed | cut -f 1,2,3 | diff - E9_hyper.bedtools.bed > /dev/null 2>&1
5  error=$?
6  if [ $error -eq 2 ]
7  then
8      echo "Error in diff command"
9  elif [ $error -eq 1 ]
10 then
11     echo "Files differ"
12 else
13     echo "OK"
14 fi
```

Output

OK