Genomic Regulation

Ivó Hernández, Helena Liz, Diego Fuentes, Fernando Freire

March 12, 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
# Obtain files for test
cd files/tracks
cat Monocyte1_11_Master_11_segments.bed | grep 'chr16' | grep 'E9' | sort -k1 -k2,3n | head
 ↪  -n 20 > monocyte1_segments.bed
cat Monocyte2_11_Master_11_segments.bed | grep 'chr16' | grep 'E9' | sort -k1 -k2,3n | head
 ↪  -n 20 > monocyte2_segments.bed
echo "Monocyte 1 segments.bed"
cat monocyte1_segments.bed
wc -l monocyte1_segments.bed
echo "Monocyte 2 segments.bed"
cat monocyte2_segments.bed
wc -l monocyte2_segments.bed
```

Output

```
Monocyte 1 segments.bed
chr16       1019400       1019600       E9
chr16       10350600       10351000       E9
chr16       10603400       10604000       E9
chr16       10608200       10608600       E9
chr16       10615400       10616000       E9
chr16       10665800       10666200       E9
chr16       10762200       10762600       E9
chr16       10830800       10831200       E9
chr16       11047800       11048400       E9
chr16       1105000       1105400       E9
chr16       11057800       11058400       E9
chr16       1107200       1107800       E9
chr16       11489600       11490600       E9
chr16       11499400       11499800       E9
chr16       11501000       11501800       E9
chr16       115200       116000       E9
chr16       11626000       11626800       E9
chr16       11890800       11891200       E9
chr16       11948000       11948200       E9
chr16       12161600       12162200       E9
      20 monocyte1_segments.bed
Monocyte 2 segments.bed
chr16       10349800       10351600       E9
chr16       10408600       10409000       E9
chr16       10603400       10604000       E9
```

```
chr16       10604800        10605200        E9
chr16       10607600        10609000        E9
chr16       10614200        10616000        E9
chr16       10762000        10763400        E9
chr16       10830800        10831200        E9
chr16       10926000        10926400        E9
chr16       11047200        11049000        E9
chr16       1105000        1105600        E9
chr16       11058000        11058600        E9
chr16       11064600        11066200        E9
chr16       1107200        1108200        E9
chr16       11109000        11109200        E9
chr16       11311600        11312800        E9
chr16       11351200        11351800        E9
chr16       11377800        11378400        E9
chr16       11450400        11451400        E9
chr16       11465600        11466400        E9
      20 monocyte2_segments.bed
```

Script 1.1.2 (text)

```
%%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
head -n 20 E9.bedtools.bed

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)

```
%%writefile files/test_tracks/bed1.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
```

Output

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

Script 1.1.4 (text)

```
%%writefile files/test_tracks/bed2.bed
chr16        60400        61400        E9
chr16        72400        72800        E9
chr16        115000        116400        E9
chr16        146600        147400        E9
chr16        146610        147400        E8
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       596700      597400      E8
chr16       596700      597400      E8
chr16       627800      630600      E9
chr16       660800      661800      E9
chr16       710800      711800      E9
chr16       717400      718200      E9
chr16       735800      736800      E9
```

Output

```
Writing files/test_tracks/bed2.bed
```

Script 1.1.5 (text)

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

Output

```
Overwriting files/test_tracks/dnase1.peaks.bed
```

1.1.2 Methods

Script 1.1.6 (python)

```python
import re as re
import subprocess

def head(path, filename, lines=20):
    """
    """
    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_uniq_features(path, filename, output_filename, sep='\t', sep2='+'):
    """
    Returns the features whitout duplicates
    """
    i = 0
    file = open(path + "/" + filename, "r")
    output_file = open(path + "/" + output_filename, "w")
    for line in file:
        chrom , output_start, output_end, segment_feature = get_parts(line)
        features = segment_feature.split("+")
        output_feature = sep2.join(list(set(features)))
        output_segment = concat_parts(chrom, output_start, output_end, output_feature)
        output_file.write(output_segment)
    file.close()
    output_file.close()

```

```python
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

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()
            # Annotate mode: save f1, advance f1, advance f2
            else:
                if output_feature == "as_file2":
                    feature = f1_feature + "+" + f2_feature
                elif output_feature != "":
                    feature = f1_feature + "+" + output_feature
                else:
                    feature = f1_feature
                if drop_feature_threshold < f1_segment_end - f1_segment_start:
                    output_segment = concat_parts(chrom, f1_segment_start, f1_segment_end, feature)
                    output_segments.write(output_segment)
                # Advance f1
                f1_segment = f1_segments.readline()
                # Advance f2
                if f1_segment_end > f2_segment_end:
                    f2_segment = f2_segments.readline()
    while(output_mode == "annotate" and f1_segment != ""):
        output_segments.write(f1_segment)
        f1_segment = f1_segments.readline()

    f1_segments.close()
    f2_segments.close()
    output_segments.close()
```

1.1.3 Tests

Script 1.1.7 (python)

```python
TEST_PATH = "files/test_tracks"
F1_FILE_TEST = "test1.bed"
F2_FILE_TEST = "test2.bed"
FILE_OUTPUT_TEST = "test_result.bed"
STATE = "E9"
CHROM = "chr16"
SEP = '\t'
```

```python
DROP = 0

failed = 0
passed = 0
launched = 0

def create_testfile(segments, test_path, test_file):
    """
    """
    output_segments = open(test_path + "/" + test_file, "w")
    for segment in segments:
        output_segment = concat_parts(segment[0], segment[1], segment[2], segment[3])
        output_segments.write(output_segment)
    output_segments.close()

def read_testfile(test_path, test_file):
    """
    """
    file_segments = open(test_path + "/" + test_file, "r")
    segments = []
    segment = file_segments.readline()
    while(segment != ""):
        chrom, segment_start, segment_end, feature = get_parts(segment)
        segments.append([chrom, str(segment_start), str(segment_end), feature])
        segment = file_segments.readline()
    file_segments.close()
    return segments

def do_test(test_number, segments_1, segments_2, test_expected_result,
            verbose=True, mode="intersect", output_feature="output_feature",
            test_path=TEST_PATH, f1_file_test=F1_FILE_TEST, f2_file_test=F2_FILE_TEST,
            file_output_test=FILE_OUTPUT_TEST,
            chrom=CHROM, f1_feature_filter="", f2_feature_filter="",
            drop_feature_threshold=DROP, with_bedtools_check=False):
    global failed, passed, launched
    try:
        launched += 1
        create_testfile(segments_1, test_path, f1_file_test)
        create_testfile(segments_2, test_path, f2_file_test)
        intersect_bed(test_path, f1_file_test, f2_file_test, test_path,
                      file_output_test, chrom=chrom,
                      f1_feature_filter=f1_feature_filter,
                       ↪ f2_feature_filter=f2_feature_filter,
                      output_feature=output_feature,
                      sep=SEP, drop_feature_threshold=drop_feature_threshold,
                       ↪ output_mode=mode)
        if verbose: head(test_path, file_output_test, 20)
        output_segments = read_testfile(test_path, file_output_test)
        if verbose: print("Threshold", drop_feature_threshold)
        if verbose: print("Result", output_segments)
        if verbose: print("Expected result", test_expected_result)
        assert output_segments == test_expected_result, "Unexpected segments"
        if with_bedtools_check:
            # Check result agains bedtools
            bedtools_cmd = "bedtools intersect"
            if mode != "intersect" : bedtools_cmd += " -wa "
            bedtools_cmd += " -a " + test_path + "/" + f1_file_test + " -b " + test_path + "/" + f2_file_test + " > " + test_path + "/" + "bedtools_" + file_output_test
            if verbose: print(bedtools_cmd)
            p = subprocess.run(bedtools_cmd, shell = True, stdout = subprocess.PIPE)
            if verbose: print("Bedtools", p.returncode, p.stdout)
            assert p.returncode == 0 and p.stdout == b'', "Bedtools command has failed"
            if mode != "intersect" :
                bedtools_cmd = "bedtools intersect -v"
                bedtools_cmd += " -a " + test_path + "/" + f1_file_test + " -b " + test_path + "/" + f2_file_test + " >> " + test_path + "/" + "bedtools_" + file_output_test
                if verbose: print(bedtools_cmd)
                p = subprocess.run(bedtools_cmd, shell = True, stdout = subprocess.PIPE)
                if verbose: print("Bedtools", p.returncode, p.stdout)
                assert p.returncode == 0 and p.stdout == b'', "Bedtools command 2 has failed"
                diff_cmd = "cat " +  test_path + "/" + "bedtools_" + file_output_test + " | sort -k 1,3 -h | diff " + test_path + "/" + file_output_test + " -"
            else:
                diff_cmd = "diff " + test_path + "/" + file_output_test + " " + test_path + "/" + "bedtools_" + file_output_test
            if verbose: print(diff_cmd)
            p = subprocess.run(diff_cmd, shell = True, stdout = subprocess.PIPE)
            if verbose: print(diff_cmd)
            if verbose: print("Diff", p.returncode, p.stdout)
            assert p.returncode == 0 and p.stdout == b'', "Diff command has failed"
        passed += 1
        print ("Passed test %s" % (test_number))
    except AssertionError:
        print ("Failed test %s: Result:\n %s\nExpected result:\n %s\n"
               % (test_number, output_segments, test_expected_result))
        failed += 1
        exit(1)

# Test 1
segments_1 = [["chr16","0","100", "A"],
              ["chr16","200","210", "A"]]
segments_2 = [["chr16","10","50", "B"]]
test_expected_result = [["chr16","10","50", "A"]]
do_test(1, segments_1, segments_2, test_expected_result, verbose = False, with_bedtools_check = True,
       output_feature = "A")

# Test 1b
segments_1 = [["chr16","0","100", "A"],
              ["chr16","200","210", "A"]]
segments_2 = [["chr16","10","50", "B"]]
test_expected_result = [["chr16","0","100", "A"],
                        ["chr16","200","210", "A"]]
```

```python
do_test(1, segments_1, segments_2, test_expected_result, verbose = True, with_bedtools_check
 ↪  = True,
       output_feature = "", mode = "annotate")

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

# Test 3
segments_1 = [["chr16","0","100", "A"],
              ["chr16","200","210", "A"]]
segments_2 = [["chr16","10","20", "B"],
              ["chr16","30","50", "B"]]
test_expected_result = [["chr16","0","100", "A+output_feature"],
                        ["chr16","200","210", "A"]]
do_test(3, segments_1, segments_2, test_expected_result, False, "annotate")

# Test 4
segments_1 = [["chr16","0","100", "A"],
              ["chr16","200","210", "A"]]
segments_2 = [["chr16","10","20", "B"],
              ["chr16","30","50", "B"]]
test_expected_result = [["chr16","10","20", "output_feature"],
                        ["chr16","30","50", "output_feature"]]
do_test(4, segments_1, segments_2, test_expected_result, False, "intersect")

# Test 5
segments_1 = [["chr16","0","100", "A"],
              ["chr16","200","210", "A"]]
segments_2 = []
test_expected_result = []
do_test(5, segments_1, segments_2, test_expected_result, False, "intersect")

# Test 6
segments_1 = [["chr16","0","100", "A"],
              ["chr16","200","210", "A"]]
segments_2 = []
test_expected_result = segments_1
do_test(6, segments_1, segments_2, test_expected_result, False, "annotate")

# Test 7
segments_1 = []
segments_2 = []
test_expected_result = segments_1
do_test(7, segments_1, segments_2, test_expected_result, False, "annotate")

# Test 8
segments_1 = []
segments_2 = []
test_expected_result = segments_1
do_test(8, segments_1, segments_2, test_expected_result, False, "intersect")
```

```python

# Test 9
segments_1 = [["chr8","0","100", "A"],
              ["chr8","100","150", "A"],
              ["chr16","200","210", "A"],
              ["chr16","300","1000", "A"]]
segments_2 = [["chr16","50","500", "B"],
              ["chr16","600","800", "B"]]
test_expected_result = [["chr16","200","210", "A+output_feature"],
                        ["chr16","300","1000", "A+output_feature"]]
do_test(9, segments_1, segments_2, test_expected_result, False, "annotate")

# Test 10
segments_1 = [["chr8","0","100", "A"],
              ["chr8","100","150", "A"],
              ["chr16","200","210", "A"],
              ["chr16","300","1000", "A"]]
segments_2 = [["chr16","50","500", "B"],
              ["chr16","600","800", "B"]]
test_expected_result = [["chr16","200","210", "output_feature"],
                        ["chr16","300","500", "output_feature"],
                        ["chr16","600","800", "output_feature"]]
do_test(10, segments_1, segments_2, test_expected_result, False, "intersect")

# Test 11
test_expected_result = [["chr16","300","500", "output_feature"],
                        ["chr16","600","800", "output_feature"]]
do_test(11, segments_1, segments_2, test_expected_result, False, "intersect",
 ↪  drop_feature_threshold=30)

# Test 12
segments_1 = [["chr8","0","100", "A"],
              ["chr8","100","150", "A"],
              ["chr16","200","300", "A"],
              ["chr16","210","320", "A"]]
segments_2 = [["chr16","210","250", "B"],
              ["chr16","305","320", "B"]]
test_expected_result = [["chr16","210","250", "output_feature"],
                        ["chr16","305","320", "output_feature"]]
do_test(12, segments_1, segments_2, test_expected_result, False, "intersect")

# Test 13
segments_1 = [["chr16","0","100", "E9"],
              ["chr16","200","300", "E9"]]
segments_2 = [["chr16","0","1000", "bbb"]]
test_expected_result = [["chr16","0","100", "E9+genes"],
                        ["chr16","200","300", "E9+genes"]]
do_test(13, segments_1, segments_2, test_expected_result, False, "annotate",
 ↪  output_feature="genes")

# Test 14
segments_1 = [["chr16","4033000","4033400", "E9"],
               ["chr16","4042200","4042600", "E9"],
               ["chr16","4250200","4250400", "E9"]]

segments_2 = [["chr16","4012649","4166186", "bbb"],
               ["chr16", "4239374", "4292081", "bbb"]]
test_expected_result = [['chr16', '4033000', '4033400', 'E9+genes'],
                         ['chr16', '4042200', '4042600', 'E9+genes'],
                         ['chr16', '4250200', '4250400', 'E9+genes']]
do_test(14, segments_1, segments_2, test_expected_result, True, "annotate",
↪  output_feature="genes")

print(" ")
if launched == passed: print("Passed All %s Test" %(passed))
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
Passed test 13
chr16        4033000        4033400        E9+genes
chr16        4042200        4042600        E9+genes
chr16        4250200        4250400        E9+genes
Threshold 0
```

```
Result [['chr16', '4033000', '4033400', 'E9+genes'], ['chr16', '4042200', '4042600',
↪  'E9+genes'], ['chr16', '4250200', '4250400', 'E9+genes']]
Expected result [['chr16', '4033000', '4033400', 'E9+genes'], ['chr16', '4042200', '4042600',
↪  'E9+genes'], ['chr16', '4250200', '4250400', 'E9+genes']]
Passed test 14

Passed All 15 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
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
cd files/tracks
```

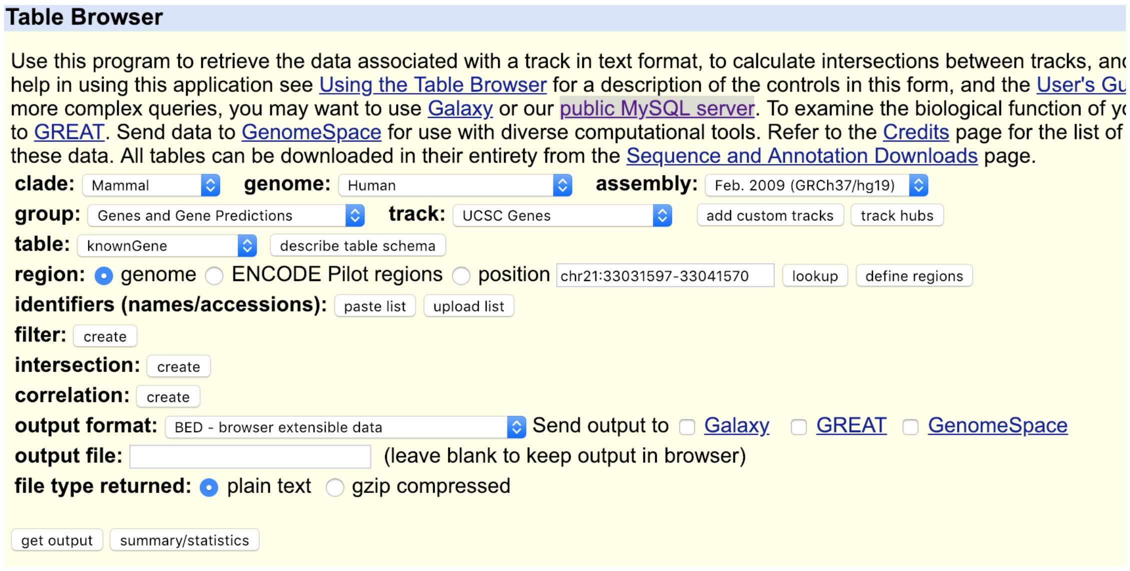


Table Browser

Use this program to retrieve the data associated with a track in text format, to calculate intersections between tracks, anc
help in using this application see Using the Table Browser for a description of the controls in this form, and the User's Gu
more complex queries, you may want to use Galaxy or our public MySQL server. To examine the biological function of yo
to GREAT. Send data to GenomeSpace for use with diverse computational tools. Refer to the Credits page for the list of
these data. All tables can be downloaded in their entirety from the Sequence and Annotation Downloads page.

clade: Mammal **genome:** Human **assembly:** Feb. 2009 (GRCh37/hg19)
group: Genes and Gene Predictions **track:** UCSC Genes
table: knownGene
region: [✓ genome] ENCODE Pilot regions position chr21:33031597-33041570
identifiers (names/accessions):
filter:
intersection:
correlation:
output format: BED - browser extensible data Send output to Galaxy GREAT GenomeSpace
output file: (leave blank to keep output in browser)
file type returned: [✓ plain text] gzip compressed

Figure 1: Table browser

```
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,
                  f1_feature_filter = "", f2_feature_filter="",
                  output_feature = OUTPUT_FEATURE, sep=SEP,
                  drop_feature_threshold = 0, 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:",
      overlap_segment_count * 100 / total_segment_count)
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

Script 1.2.2 (text)

```
%%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 + "_genes.bed"
OUTPUT_FILE = STATE + "_" + OUTPUT_FEATURE + "_genes.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:",
      overlap_segment_count * 100 / total_segment_count)
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+exons
chr16       629000      629400      E9+genes+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_genes.bed
```

1.2.4 Annotate upstream 200 overlap

Script 1.2.4 (python)

```python
OUTPUT_FEATURE = "up200"
INPUT_FILE = STATE + "_exons_genes.bed"
OUTPUT_FILE = STATE + "_" + OUTPUT_FEATURE + "_exons_genes.bed"
ANNOTATION_TRACK = "hg19_up200_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:",
      overlap_segment_count * 100 / total_segment_count)
print("Output file:", OUTPUT_FILE)
```

Output

```
chr16       60400       61400       E9+up200
chr16       72600       72800       E9+up200
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+exons
chr16       629000       629400       E9+genes+exons
```

```
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_genes.bed
```

1.2.5 Is the CTTFB intergenic?

Script 1.2.5 (text)

```
%%bash
cd files/tracks
echo "Count of E9 segments contained completely in gene segments"
bedtools annotate -i E9.bed -files hg19_genes_sorted.bed | grep -c '1.000000'
echo "Count of E9 segments out of gene segments"
bedtools annotate -i E9.bed -files hg19_genes_sorted.bed | grep -c '0.000000'
echo "Count of E9 segments overlaping gene segments"
bedtools annotate -i E9.bed -files hg19_genes_sorted.bed | grep -v '0.000000' | grep -c -v
↪ '1.00000'
#grep 'hypo' E9_hypo_hyper.bed | cut -f 1,2,3 | diff - E9_hypo.bedtools.bed > /dev/null 2>&1
echo "Some intergenic segments"
bedtools annotate -i E9.bed -files hg19_genes_sorted.bed | grep '0.000000' | head
echo "Some intragenic segments"
bedtools annotate -i E9.bed -files hg19_genes_sorted.bed | grep '1.000000' | head

echo "Monocyte 1 only to verify a possibly error"
echo "Count of E9 segments contained completely in gene segments"
bedtools annotate -i monocyte1_segments.bed -files hg19_genes_sorted.bed | grep -c '1.000000'
echo "Count of E9 segments out of gene segments"
bedtools annotate -i monocyte1_segments.bed -files hg19_genes_sorted.bed | grep -c '0.000000'
echo "Count of E9 segments overlaping gene segments"
bedtools annotate -i monocyte1_segments.bed -files hg19_genes_sorted.bed | grep -v
↪ '0.000000' | grep -c -v '1.00000'
```

Output

```
Count of E9 segments contained completely in gene segments
208
Count of E9 segments out of gene segments
248
Count of E9 segments overlaping gene segments
12
Some intergenic segments
chr16        8617600        8618200        E9        0.000000
chr16        8749000        8749400        E9        0.000000
chr16        11501000        11501800        E9        0.000000
chr16        14991000        14991400        E9        0.000000
chr16        24264200        24265200        E9        0.000000
chr16        28081800        28082600        E9        0.000000
chr16        28229400        28229800        E9        0.000000
chr16        78036800        78037200        E9        0.000000
```

```
chr16       82263200        82264200        E9       0.000000
chr16       84655600        84657000        E9       0.000000
Some intragenic segments
chr16       12188400        12190000        E9       1.000000
chr16       57933400        57934200        E9       1.000000
chr16       81919400        81920200        E9       1.000000
chr16       835400      836200      E9       1.000000
chr16       21168000        21168400        E9       1.000000
chr16       49626800        49627800        E9       1.000000
chr16       71401200        71401600        E9       1.000000
chr16       87883400        87883800        E9       1.000000
chr16       115200      116000      E9       1.000000
chr16       146600      147400      E9       1.000000
Monocyte 1 only to verify a possibly error
Count of E9 segments contained completely in gene segments
285
Count of E9 segments out of gene segments
348
Count of E9 segments overlaping gene segments
13
```

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).

Script 1.3.2 (python)

```python
intersect_bed(PATH, STATE + ".bed", "wgEncodeOpenChromDnaseMonocd14Pk.narrowPeak.bed",
                  PATH, STATE + "_dnase1.bed", chrom = CHROM,
                  f1_feature_filter = "", f2_feature_filter = "",
                  output_feature = STATE + "_dnase1_overlap", sep = SEP,
                  drop_feature_threshold = 10)
head(PATH, STATE + "_dnase1.bed", 10)
coverage_peaks = bed_coverage(PATH, "wgEncodeOpenChromDnaseMonocd14Pk.narrowPeak.bed",
 ↪  sep='\t')
coverage_state = bed_coverage(PATH, STATE + ".bed", sep='\t')
print("")
print("Coverage DNASE peaks:", coverage_peaks, "bps")
print("Coverage E9:", coverage_state, "bps")
print("Percent overlap over total coverage peaks:", coverage_state * 100 / coverage_peaks)
print("Output file:", STATE + "_dnase1.bed")
```

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
OUTPUT_FEATURE = "dnase1"

intersect_bed(PATH, STATE + ".bed", "wgEncodeOpenChromDnaseMonocd14Pk.narrowPeak.bed",
                  PATH, STATE + "_dnase1.bed", chrom = CHROM,
                  f1_feature_filter = STATE, f2_feature_filter="",
                  output_feature = OUTPUT_FEATURE, sep=SEP,
                  drop_feature_threshold = 10, output_mode="annotate")
head(PATH, STATE + "_dnase1.bed", 10)

overlap_segment_count = bed_segment_count_by_re_feature(PATH, STATE + "_dnase1.bed",
```

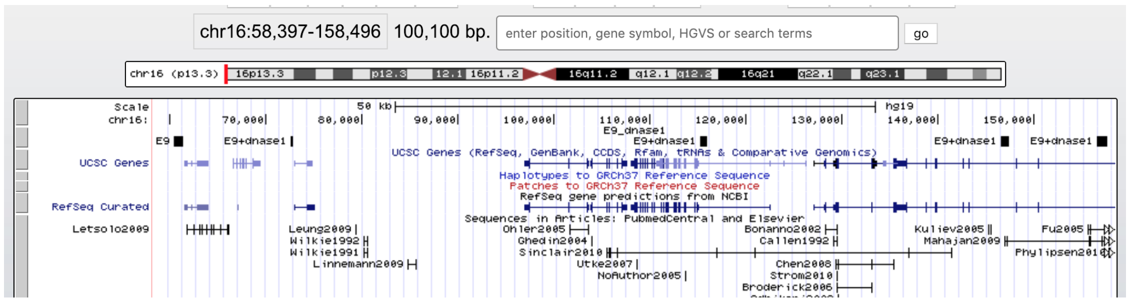


Figure 2

```
                                                                OUTPUT_FEATURE)
print("")
print("Count of state segments overlapped:", overlap_segment_count)
total_segment_count = bed_segment_count_by_re_feature(PATH, STATE + "_dnase1.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 + "_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)

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

Visual inspection

Script 1.3.5 (text)

```
%%bash
export TRACKS=files/tracks/
echo "Counts"
wc -l ${TRACKS}E9_dnase1.bed
wc -l ${TRACKS}E9.bed
tail ${TRACKS}E9_dnase1.bed
echo
tail ${TRACKS}E9.bed
echo
head ${TRACKS}E9_dnase1.bed
echo
head ${TRACKS}E9.bed
echo
echo "Counts of annotations:"
cat ${TRACKS}E9_dnase1.bed | grep "dnase1" | wc -l
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-

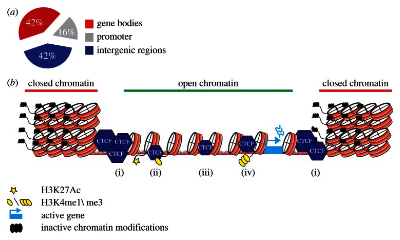


Figure 3

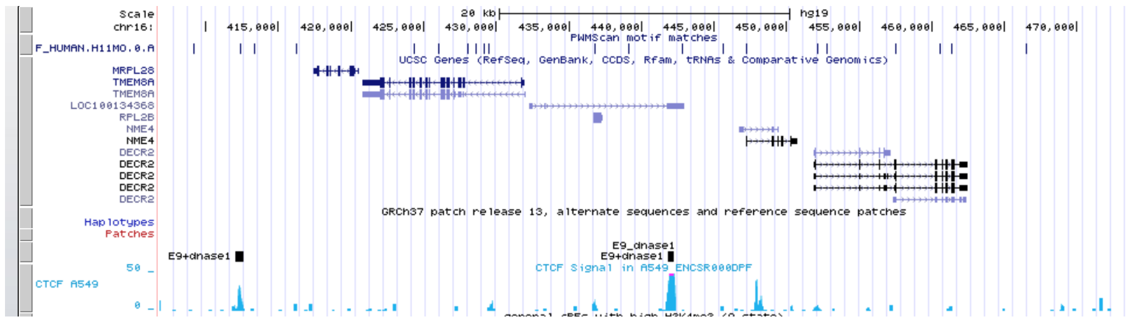


Figure 4

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.

Also it seems that CTCF regulates itself:

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.

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).

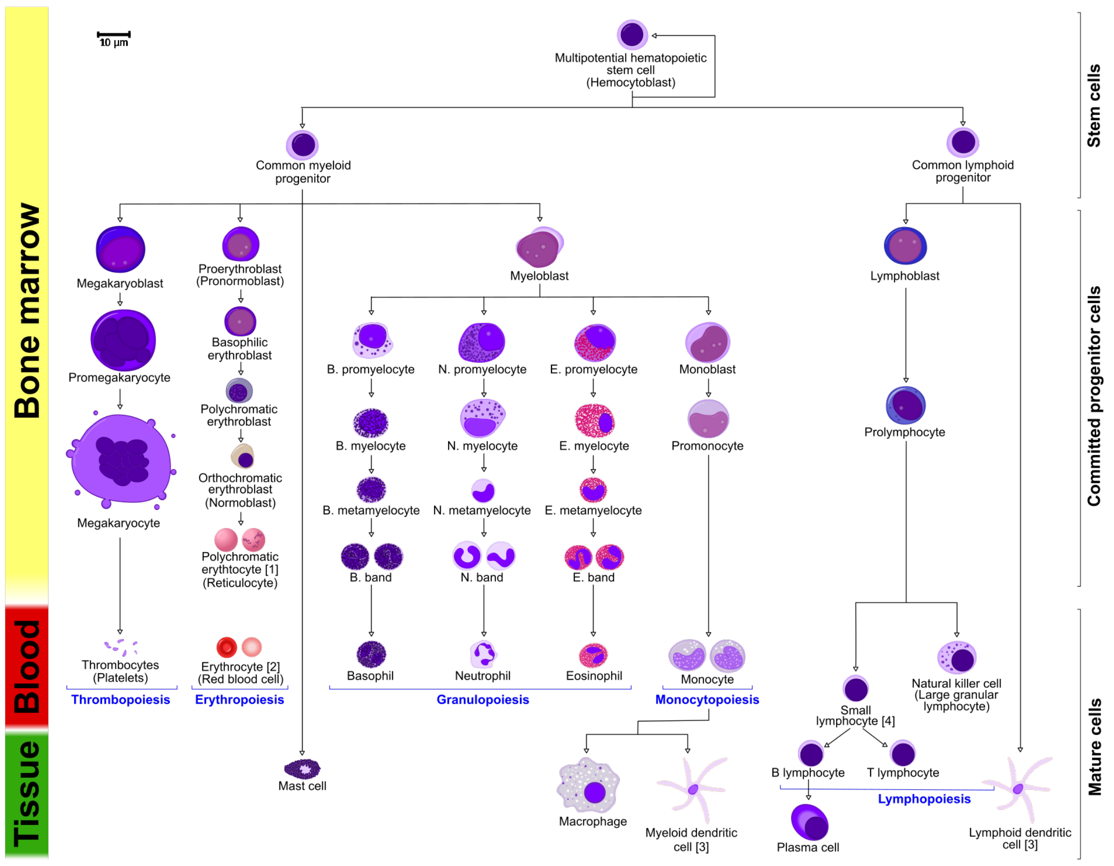


Figure 5: Human hematopoiesis

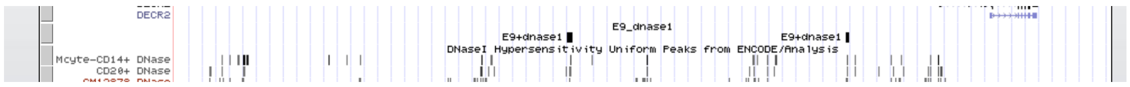


Figure 6: E9 vs DNaseI blood cell types

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.

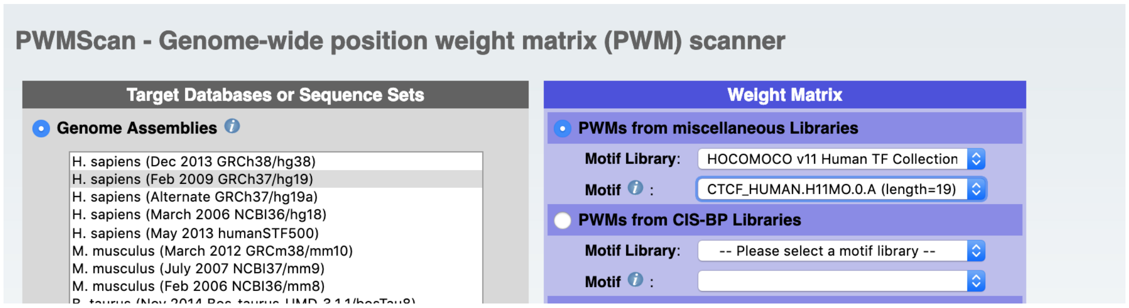
It would be interesting to find at least the consensus sequence of CTCF binding sites.

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.



PWMScan - Genome-wide position weight matrix (PWM) scanner

Target Databases or Sequence Sets

[✓ Genome Assemblies]

- H. sapiens (Dec 2013 GRCh38/hg38)
- H. sapiens (Feb 2009 GRCh37/hg19)
- H. sapiens (Alternate GRCh37/hg19a)
- H. sapiens (March 2006 NCBI36/hg18)
- H. sapiens (May 2013 humanSTF500)
- M. musculus (March 2012 GRCm38/mm10)
- M. musculus (July 2007 NCBI37/mm9)
- M. musculus (Feb 2006 NCBI36/mm8)

Weight Matrix

[✓ PWMs from miscellaneous Libraries]

Motif Library: HOCOMOCO v11 Human TF Collection

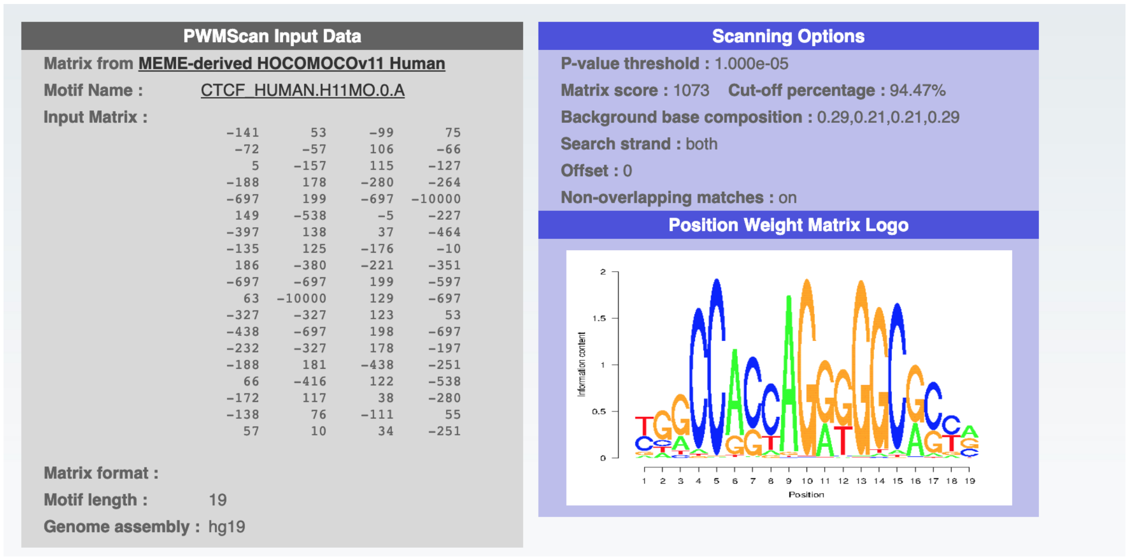
Motif : CTCF_HUMAN.H11MO.0.A (length=19)

PWMs from CIS-BP Libraries

Motif Library: -- Please select a motif library --

Motif :

Figure 7



PWMScan Input Data

Matrix from MEME-derived HOCOMOCOv11 Human

Motif Name : CTCF_HUMAN.H11MO.0.A

Input Matrix :

```
 -141      53     -99      75
  -72     -57     106     -66
    5    -157     115    -127
 -188     178    -280    -264
 -697     199    -697  -10000
  149    -538      -5    -227
 -397     138      37    -464
 -135     125    -176     -10
  186    -380    -221    -351
 -697    -697     199    -597
   63  -10000     129    -697
 -327    -327     123      53
 -438    -697     198    -697
 -232    -327     178    -197
 -188     181    -438    -251
   66    -416     122    -538
 -172     117      38    -280
 -138      76    -111      55
   57      10      34    -251
```

Matrix format :

Motif length : 19

Genome assembly : hg19

Scanning Options

P-value threshold : 1.000e-05

Matrix score : 1073 Cut-off percentage : 94.47%

Background base composition : 0.29,0.21,0.21,0.29

Search strand : both

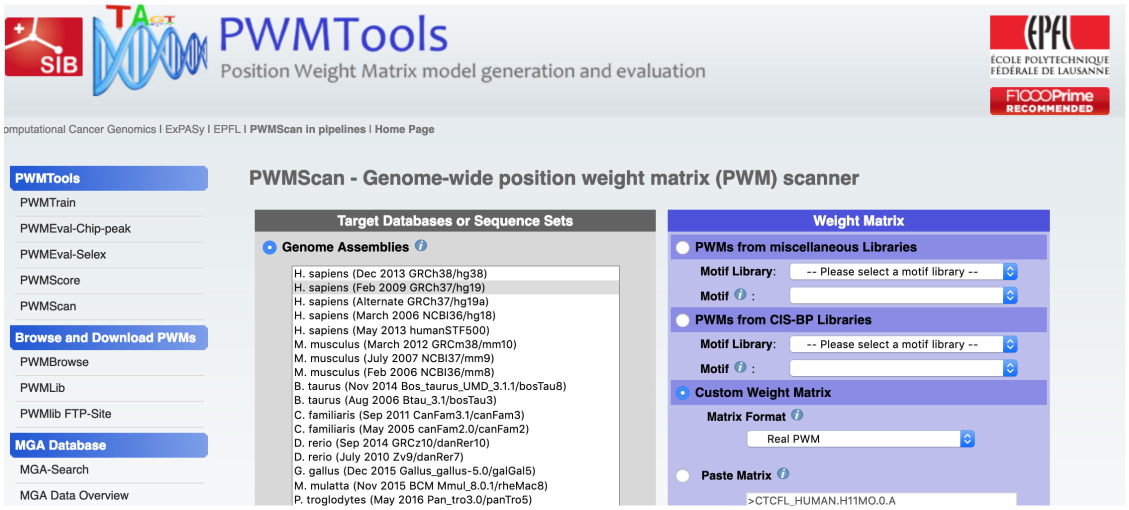
Offset : 0

Non-overlapping matches : on

Position Weight Matrix Logo

Figure 8

1.5.2 Search for CTCF binding sites



PWMScan - Genome-wide position weight matrix (PWM) scanner

Target Databases or Sequence Sets

[✓ Genome Assemblies]

- H. sapiens (Dec 2013 GRCh38/hg38)
- H. sapiens (Feb 2009 GRCh37/hg19)
- H. sapiens (Alternate GRCh37/hg19a)
- H. sapiens (March 2006 NCBI36/hg18)
- H. sapiens (May 2013 humanSTF500)
- M. musculus (March 2012 GRCm38/mm10)
- M. musculus (July 2007 NCBI37/mm9)
- M. musculus (Feb 2006 NCBI36/mm8)
- B. taurus (Nov 2014 Bos_taurus_UMD_3.1.1/bosTau8)
- B. taurus (Aug 2006 Btau_3.1/bosTau3)
- C. familiaris (Sep 2011 CanFam3.1/canFam3)
- C. familiaris (May 2005 canFam2.0/canFam2)
- D. rerio (Sep 2014 GRCz10/danRer10)
- D. rerio (July 2010 Zv9/danRer7)
- G. gallus (Dec 2015 Gallus_gallus-5.0/galGal5)
- M. mulatta (Nov 2015 BCM Mmul_8.0.1/rheMac8)
- P. troglodytes (May 2016 Pan_tro3.0/panTro5)

Weight Matrix

PWMs from miscellaneous Libraries

Motif Library: -- Please select a motif library --

Motif :

PWMs from CIS-BP Libraries

Motif Library: -- Please select a motif library --

Motif :

[✓ Custom Weight Matrix]

Matrix Format: Real PWM

Paste Matrix

>CTCFL_HUMAN.H11MO.0.A

Home page:

Select for CTCF PWM:

Logo for CTCFB:

Download bed over hg19a (there is no option for hg19)
Overlaps with bedtools annotate (could be done with our SW):
E9 segments (monocyte1 and monocyte 2) with overlap:

```
bedtools annotate -i E9.bed -files hg19_ctcfb_sorted.bed | grep -v '0.000000' | wc -l
> 380
```

So 380 segments from 468 E9 segments: 81% We expect more 100% close value, but the base experiment of the CTCF binding sites was over A549 lung cell line.

If we trust both experiments, sources of both tracks, we could think that CTCF binding sites would be very similar across cell-lines.

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)

```
%%bash
cd files/tracks
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
grep 'hypo' E9_hypo_hyper.bed | cut -f 1,2,3 | diff - E9_hypo.bedtools.bed > /dev/null 2>&1
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
```

Script 1.6.5 (text)

```
%%bash
cd files/tracks
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
grep 'hyper' E9_hypo_hyper.bed | cut -f 1,2,3 | diff - E9_hyper.bedtools.bed > /dev/null 2>&1
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
```

Script 1.6.6 (python)

```
OUTPUT_FEATURE = "as_file2"
OUTPUT_FILE = STATE + "_hypo_hyper_dnase1.bed"
INPUT_FILE =  "E9_dnase1.bed"
intersect_bed(PATH, STATE + "_hypo_hyper.bed", INPUT_FILE,
                  PATH, OUTPUT_FILE, chrom = CHROM,
                  f1_feature_filter = "", f2_feature_filter="",
                  output_feature = OUTPUT_FEATURE, sep=SEP,
                  drop_feature_threshold = 0, output_mode="annotate")
head(PATH, OUTPUT_FILE, 10)

overlap_segment_count = bed_segment_count_by_re_feature(PATH, OUTPUT_FILE,
                                                        OUTPUT_FEATURE)
print("Output file:", OUTPUT_FILE)

bed_uniq_features(PATH, OUTPUT_FILE, "E9_hypo_hyper_dnase1.unif.bed", sep='\t', sep2='+')
head(PATH, "E9_hypo_hyper_dnase1.unif.bed", 10)
```

Output

```
chr16	60400	61400	E9+hypo+hyper+E9
chr16	72600	72800	E9+hyper+E9+dnase1
chr16	115200	116000	E9+hypo+hyper+E9+dnase1
chr16	146600	147400	E9+hypo+hyper+E9+dnase1
chr16	156600	157600	E9+hyper+E9+dnase1
chr16	167800	168200	E9+hypo+hyper+E9+dnase1
chr16	412000	412600	E9+hypo+hyper+E9+dnase1
chr16	441800	442200	E9+hyper+E9+dnase1
```

```
chr16        537600        538000        E9+hyper+E9+dnase1
chr16        597000        597400        E9+E9+dnase1
chr16        629000        629400        E9+hypo+E9+dnase1
Output file: E9_hypo_hyper_dnase1.bed
chr16        60400        61400        E9+hypo+hyper
chr16        72600        72800        E9+dnase1+hyper
chr16        115200        116000        E9+dnase1+hypo+hyper
chr16        146600        147400        E9+dnase1+hypo+hyper
chr16        156600        157600        E9+dnase1+hyper
chr16        167800        168200        E9+dnase1+hypo+hyper
chr16        412000        412600        E9+dnase1+hypo+hyper
chr16        441800        442200        E9+dnase1+hyper
chr16        537600        538000        E9+dnase1+hyper
chr16        597000        597400        E9+dnase1
chr16        629000        629400        E9+hypo+dnase1
```