A Report on the “GapFilling in Genome Assembly” Project

Read length = 100
No. of Reads = 300,000
Reference Genome Length = 1,091,731
Total_num of Iterations in EM= 5

For this debugging, we have coverage = 100*300,000/1091731 == 27

Here is the code structure:

```
1. For each gap_estimate from [Gap_Min to Gap_Max]
2.         Compute the likelihood by placing the reads[from gaps_0.sam] into that  gap_estimate
3.         Store the max likelihood coming from placeread() function
4. Finally Fill the gap with the gap_estimate with highest likelihood calling Finalize() function
```

Observation - 1:

The Finalize() function based on the final(best) gap_estimate at Line 4, re-computes the counts_gap array and fills the gap based on that 2D array. ***But sometimes, when we send the wrong_value of gap_estimate to this function***, the probability of reads(in the gap) using which the counts_gap array will be initialized doesn't match the cut-off value computed by the ComputeProbabilty() function. So, all/most of the reads are discarded and counts_gap array is uninitialized and the gap is filled with all A’s like “AAAAAAAAAAAAAAAAA” based on the specific code.

Observation - 2:

Why do we send incorrect value to the Finalize() function?

It’s because, sometimes, in the for loop of line 1, the likelihood values that come from placeread() function is all negative infinity(- INF = - DBL_MAX). So, the best gap_estimate becomes the Gap_max or Gap_min [whether we want to update or not]. EIther one is incorrect. So, in those cases, the Edit Distance is huge.

Observation - 3:

The most interesting finding that I have came across is that the code can find the perfect gap_string no matter how good/bad the reads are, if we can correctly guess the gap_length a.k.a. Gap_estimate. So, it’s important that the likelihood values don’t become -INF and also stop just at the correct point. So, when the likelihood value is maximum for the gap_estimate == Actual gap_length, the ED always becomes 0.

Observation - 4:

Why is the likelihood returned by placeRead() function infinity?

In placeRead() the structure is following:

```
1. For each read in the gap
2.      Compute the probability for that gap based on the following formula:
        temp_prob *= (probsGap[index][charCode]*(1-errorPosDist[readIndex])
                     + errorPosDist[readIndex]*errorProbsGap[index][charCode]);
3.      Add those log(temp_prob) cumulatively
4.Return the final_sum
```

So, it becomes:

```
        max_likelihood += log(probabilities) = log(p1) + log(p2) + log(p3)+...
                                             =log(p1*p2*p3...)
        Return max_likelihood
```

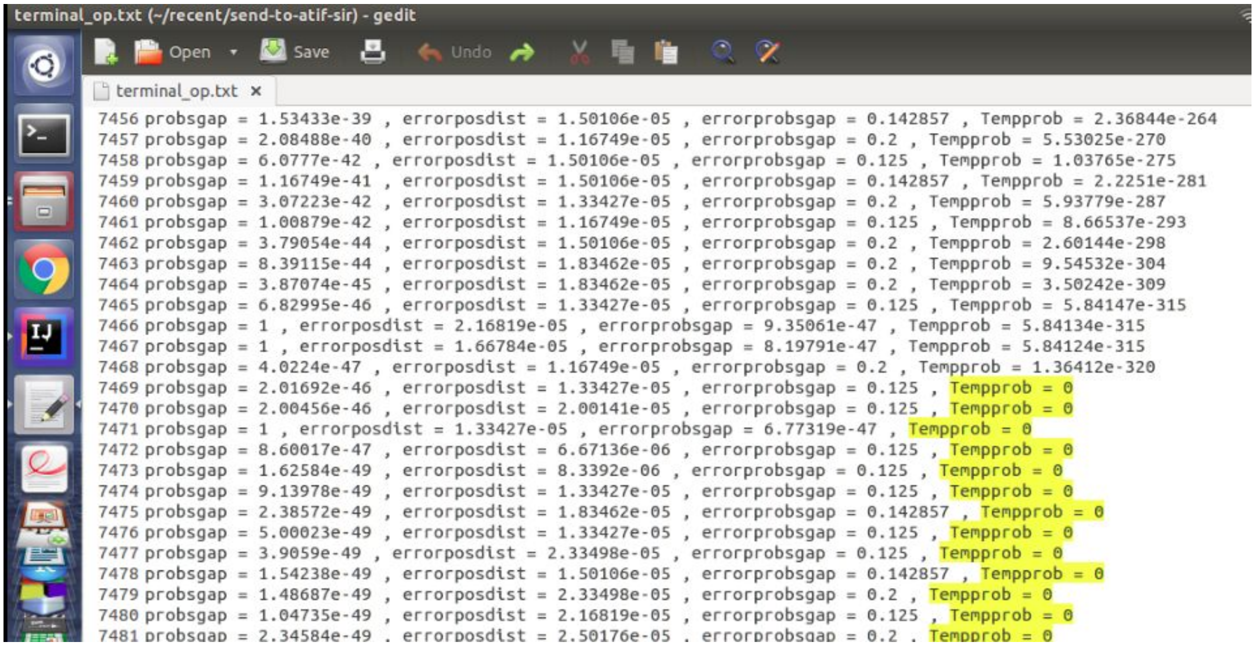
But for some reads, these temp_probs[p1, p2, p3 etc.] becomes 0 and so, the result becomes log(0) and it returns -INF.

Observation - 5:

Why temp_prob becomes 0?

Last time we met you, we thought that for some indices during calculation, the **probs_gap value** and **errorprobs_gap value** becomes 0 at the same time, causing the temp_prob to be 0 in line 2.

But I checked it thoroughly multiple times within appropriate index region of the calculation that, ***they are mutually exclusive[initially]*** and never becomes 0 at the same time/calculation. The reason why it becomes 0 can be understood from the following screenshot:



```
probsgap = 1.53433e-39 , errorposdist = 1.50106e-05 , errorprobsgap = 0.142857 , Tempprob = 2.36844e-264
probsgap = 2.08488e-40 , errorposdist = 1.16749e-05 , errorprobsgap = 0.2 , Tempprob = 5.53025e-270
probsgap = 6.0777e-42 , errorposdist = 1.50106e-05 , errorprobsgap = 0.125 , Tempprob = 1.03765e-275
probsgap = 1.16749e-41 , errorposdist = 1.50106e-05 , errorprobsgap = 0.142857 , Tempprob = 2.2251e-281
probsgap = 3.07223e-42 , errorposdist = 1.33427e-05 , errorprobsgap = 0.2 , Tempprob = 5.93779e-287
probsgap = 1.00879e-42 , errorposdist = 1.16749e-05 , errorprobsgap = 0.125 , Tempprob = 8.66537e-293
probsgap = 3.79054e-44 , errorposdist = 1.50106e-05 , errorprobsgap = 0.2 , Tempprob = 2.60144e-298
probsgap = 8.39115e-44 , errorposdist = 1.83462e-05 , errorprobsgap = 0.2 , Tempprob = 9.54532e-304
probsgap = 3.87074e-45 , errorposdist = 1.83462e-05 , errorprobsgap = 0.2 , Tempprob = 3.50242e-309
probsgap = 6.82995e-46 , errorposdist = 1.33427e-05 , errorprobsgap = 0.125 , Tempprob = 5.84147e-315
probsgap = 1 , errorposdist = 2.16819e-05 , errorprobsgap = 9.35061e-47 , Tempprob = 5.84134e-315
probsgap = 1 , errorposdist = 1.66784e-05 , errorprobsgap = 8.19791e-47 , Tempprob = 5.84124e-315
probsgap = 4.0224e-47 , errorposdist = 1.16749e-05 , errorprobsgap = 0.2 , Tempprob = 1.36412e-320
probsgap = 2.01692e-46 , errorposdist = 1.33427e-05 , errorprobsgap = 0.125 , Tempprob = 0
probsgap = 2.00456e-46 , errorposdist = 2.00141e-05 , errorprobsgap = 0.125 , Tempprob = 0
probsgap = 1 , errorposdist = 1.33427e-05 , errorprobsgap = 6.77319e-47 , Tempprob = 0
probsgap = 8.60017e-47 , errorposdist = 6.67136e-06 , errorprobsgap = 0.125 , Tempprob = 0
probsgap = 1.62584e-49 , errorposdist = 8.3392e-06 , errorprobsgap = 0.125 , Tempprob = 0
probsgap = 9.13978e-49 , errorposdist = 1.33427e-05 , errorprobsgap = 0.125 , Tempprob = 0
probsgap = 2.38572e-49 , errorposdist = 1.83462e-05 , errorprobsgap = 0.142857 , Tempprob = 0
probsgap = 5.00023e-49 , errorposdist = 1.33427e-05 , errorprobsgap = 0.125 , Tempprob = 0
probsgap = 3.9059e-49 , errorposdist = 2.33498e-05 , errorprobsgap = 0.125 , Tempprob = 0
probsgap = 1.54238e-49 , errorposdist = 1.50106e-05 , errorprobsgap = 0.142857 , Tempprob = 0
probsgap = 1.48687e-49 , errorposdist = 2.33498e-05 , errorprobsgap = 0.2 , Tempprob = 0
probsgap = 1.04735e-49 , errorposdist = 2.16819e-05 , errorprobsgap = 0.125 , Tempprob = 0
probsgap = 2.34584e-49 , errorposdist = 2.50176e-05 , errorprobsgap = 0.2 , Tempprob = 0
```

The reason is that, because of the nature of the small values of the 2D arrays, at each iteration the values become smaller and smaller after multiplication and eventually becomes 0.

May be, as the probability of A in a gap position becomes close to 1, the corresponding error probability of A on that gap becomes close to 0, so they are not mutually exclusive anymore after some iterations.

Observation - 6:

I tried to understand **the reason why the values are so small**. I found out that the effect is a bit of “Vicious Circle” property.

Initially, the probsgap = 0.25 0.25 0.25 0.25
At every iteration of EM-step, the following is occurring:

```
While(num_itr <=5)
{
    1. Calculate the probabilities(temp_prob) using initial/new probs_gap array and fill the array counts_gap with these small probabilities.
    2. Update the  probs_gap based on the new counts_gap array
    3. Goto Step - 1
}
```

So, the probs_gap array keeps getting smaller because of the smaller probabilities and the probabilities keeps getting smaller because of the smaller probs_gap array value.

It’s not the values become -INF instantly at first iteration. After 3 or 4 (Random) iterations, the probability gradually diminish to 0.

Solution:

As, I could not find a way to control the diminishing values of arrays and probabilities, I checked for other ways.

1) **Sol - 1 :** Simply instead of taking the log probabilities, if we sum them and return the log value, the code works perfectly fine. For example:

```
        max_likelihood += probabilities =(p1) +(p2) +(p3)+...
        Return log(max_likelihood)
```

So, the 0 probabilities have no effect here. But, according to theory we should multiply the probabilities for independent events, not add them. So, I don’t know why the code works so well for this case. Everytime the likelihood is maximum for gap_estimate == gap_legth

2) **Sol - 2 :** Also, we can ignore the probabilities =0 and continue with rest like:

```
max_likelihood += log(probabilities) = log(p1) + log(p3)+...
                                     =log(p1*p3...) [if P2 == 0]
Return max_likelihood
```

Which also gives similar results [Yet to test]

3. Sol - 3:

I tried to store the best likelihood value among all iterations in EM step and use that value as likelihood estimate for a particular gap length. But it gives poor results.

Alternative Solution:

If we don’t want to change the log calculations, we can still get better results if we compute the sequence based on the **probsGap** array instead of the **countsGap** array.

Pros:

1. It bypasses the Finalize() function and Computelikelihood() function[Applicable to Solution 1 too]. They become redundant.

Cons:

1. It is better than current code but does not work anywhere near as better as Solution - 1.