Genom_Proj2

May 30, 2020

1 Emre Erdem 150115501

In this project I'll present you my motif generator, DNA generator and my randomized motif search algorithm. The parameters are flexible and you can toy with it to see how they change Let me start

[983]:
```
import random as rd
import string
import timeit
```

1.0.1 Text Generation

In this segment, we have necessary functions to implement motif generation. As you can see from the output it generates the motif with the desired mutation level with given parameters (or default).

[3]:
```
def randString(length=10, endline=True): #Random dna string generator
    # put your letters in the following string
    your_letters='ATGC'
    if(endline):
        return ''.join((rd.choice(your_letters) for i in range(length)))+"\n"
    else:
        return ''.join((rd.choice(your_letters) for i in range(length)))

def randMut(oldGene):
    your_letters='atgc'
    mutate = rd.choice(your_letters)
    while (oldGene.lower() == mutate):
        mutate = rd.choice(your_letters)
    return mutate

def checkIfMutated(check = [], rand = 0):    #goes thru already mutated indexes␣
↪to check if already mutated
    for i in check:
        if(i == rand):
            return True
    return False
```

```
def generateMotif(length=10, mutation=4):          #generates a motif of given␣
↪length and given mutation
    check = [0 for i in range(mutation)]
    string = randString(length, endline=False)
    for i in range(mutation):                      #generate numbers 4 distinct index␣
↪numbers to mutate
        rand = rd.randint(0,length-1)          #generate mutation index number
        while(checkIfMutated(check, rand)): #check if the gene in given index␣
↪already mutated
            rand = rd.randint(0,length-1)   #if mutated choose another random␣
↪index
        check[i] = rand
    #print(string)
    string = list(string)                          #changes to list form to be able to␣
↪edit the string
    for i in check:
        string[i] = randMut(string[i])             #mutate the indexes. Indexes stored␣
↪in 'check'
    string = ''.join(string)
    return string

#print(generateMotif())
```

```
TcGccTCCGt
```

This segment generates the whole DNA text and inserts motifs randomly to each line. You can differentiate each line with the indentation in text. Since it can't contain 500 characters in one line in jupyter

```
[4]: def generateText(length=10, mutation=4, lineLength=500, numberOfLines=5):
         listToFill = ""
         for i in range(numberOfLines):                    #iterates through number of␣
     ↪lines
             listToFill = listToFill + randString(lineLength)  #adds another line of␣
     ↪dna string
             indexToMotif = rd.randint(i*lineLength,␣
     ↪i*lineLength+(lineLength-length)) #decides the index to put the motif to
             while (indexToMotif%lineLength == 0 and indexToMotif != 0):        ␣
     ↪#checks to make sure not to hit a '\n'
                 indexToMotif = rd.randint(i*lineLength,␣
     ↪i*lineLength+(lineLength-length)) #if it hits a '\n' randomize again
             motifToInsert = generateMotif(length, mutation)       #generates the␣
     ↪motif to insert
             listToFill = list(listToFill)
             for k in range(length): #inserts motif to the random index as the␣
     ↪length of the motif
                 listToFill[indexToMotif+k] = motifToInsert[k] #replaces char by char
```

```
        listToFill = ''.join(listToFill)
    return ''.join(listToFill)
#print(generateText())
```

```
GACGGTACTAGCCGGGCTTCTTAGCGATGGTTTATTCGACCATGGTGGTGGTCGACCACTCCGTCTCGGTGAGTTGATGG
TTGTTTAGGCATTCGCGGAACTGCACGCCGTAGCAGAGACCATAATACCTAGGTTACTTTCACTTGCTACCTTCCGACCG
ATGCGTTCGTTCCGAACAGTAGCCACTTTTATATTTCTATTTAGCCACATATTCCCCCGAGGTGTGCGGCCTAACCGGTA
GAGAATAGAGCCTTTAGAAGGCAAACtAcTTgGtCAGACGACGAGACTTACTCTGGCCACGGAGTTTACCGTACCATAAT
GCAGCGATTATTTTACCCGGGCAAGAATGCTCCAACGATATTACGCCAGGTTACACAACAATACACCAAGATTAGCGTAC
CCAGGGTTGGCGCACATTGGCTCTAGGTTGCTAGGGCAATTTAACTGCGGCGGGTTACCAAAAGGCTTTTCCAAAGTGTA
TCAGCCAGCGCTAGAGAGGT
GGGTATAGGCGActActCCCATACAACTTCGGCGTGTGGGGTTGACGGGTTGACAATATACGCAACAAGGTAAAGAGCGT
TCCTTAGCGCGAGGTATACTAATTATAAGTGTCAGTGCCGCGTTGACTCTATCGTATTTTCCCAGGAAAACTGTGCTAAA
GTAGTGGGTCCGCGAGTTATCAACGTCAAAGATGGAAACTATATCTTTTTGGGATTCTGCTGTAACATGCTGCACAGTGA
CAAGAGGCACAGCTTACAAGCTGATGTTGTTCAGGGAGCAGAGCAGTACCCATTGGGGCTGAATTATTGATTATAGCATG
AATCAGAGAATCATGTCATTATCAGTCCTCGGTGTAAAATTTTGACCCCTACGTCAGCCACTAGCGGATTCCGTAGATAA
CTCAGACGGTTGAGAATCGCAAACATTTGCGGTCGATCGCTCACATGCCAGAACCAATACGTCAGTAGCTGCCGAAGAGA
GTGCTTTTGTTCGACATTTA
TTACTCTTAGTTGAATAGCAGCGGCCGGTTTAAGGTGTCTAAGGTAATTAGTACTTGGGAAGTGCGCAAGACAATAGATA
AGCTTTCTCCTGAaTGaGgAtAGGCGCCCGGTCTTGGTGTCGCACTTCGCGCACTTCCGAATTCCATAAAGCTTTGATTG
CGATTTGACAAGCCAGTCGGATACCCGATATTTCCATGTCCCGAATCCGGTACGGAGAATAGGGTTCAAATGACCTACCT
CTTCCAGCCATTTAGTGTCTCGTGGCGTTCCGATATAACAGGCAACGTTTATTTTCGGATGGGTGACCAGATGTACTTCA
TTGATCAAGCGCCGGCGTTTCACCTGAATGTGACTTGACTCATGGTATGTTCACCCATCGAGAAGACGCGTTGGAATTCC
GAGCAGGCGGGCATCAGCAAGGCTACGCGATCACTAGTCGAGTACGGAAGGACACGAGAATGTGCCCTGCTGGTCAATTC
ACAGCCAAGTCCTGAGGCAA
ATGGGACAGTAGTATGATTGCGTCGTTTAGAGTCAACCTCGATAGCCCTCATGATGGACCAACTGGTCTAATATAAGATT
TGCCCTGTGTCCGAAAGATGGGTAATGTTCGAGTAACAGCCAGGGGATCGAGCAGCGGTAGACAAACCGGTCGACGCTCT
CCCCGTCTCAAGCAAGTGTGCTCAGGAGGGCGGGCCGTTCTGTGTGTGAGGGCACACTGGTTGGACGCCACGCACCCATG
CATCATACTCGGAAGGGTCAGACAGGAGCGTAAGTCCATCTCAGGCTTGTACTCAACAAATCGCCGATATAGAGCGACAG
GATCTTACACGGGGGGGGAGGTTCAAACAAGCAGCAAAGATACCGTTTTAGTTATTAAGTGGATTTGAAGTAGGTATTTG
CATATTATGCGGGCCCGAGGTATTAGTGTAGGTGTATTTAGGGGAGTGTCGTACCCTTCCCTGGTCATCCacAGaCcTCG
GGAAGTATCGCTATCATGGT
ATCAATTTAACCGCTAGCACGCTTCGGTAAGGCTGGAAGTGAAGTGATGATTGGTGTTCCATTCTGTTTGACATTTTGAG
TAGAAGACGACCGAGAAAGTAGGGTGCTAAGCCCAGCCCGATAACCAGGTAATACTTTCGGAATAGACTGAAACACTAGC
AGGCGATCTCCTTGGCAGCAATCTACACTTAAATATAATTGGGCTTCCGACCTTCAGAGGCCGGCACTCACAGCATGCAA
GAACGACCGATCTTTTAGAGTTTGTGACTcGcaACaAATTGTGTCGATCGATCTCCCTAATCTCAGATACAATGCATCAC
TCCTCAGTCAGACACTCCGTCTGCCCACTTTGATTGAAGGTAAGGTGATCATGGTACGGCAAAACTCAAAGGCTCATGGC
AGAAGCCGGAACGCCTAATTTGTATCAGTCCGTCGGGTACTCCGTTGTCTCAGGGAGATCAAGTACCAACACTACGATTT
AACTCTATAGGACTACAATT
```

Using 5 lines for easier implementation and taking a quick look of how it performs. Also testing for reading and writing operations

```
[5]: f = open("Generated.txt", "w")
     f.write(generateText(length=10, mutation=4, lineLength=500, numberOfLines=5))
     f.close()
     f = open("Generated.txt", "r")
```

```python
text = f.read()
f.close()
#print(text)
```

```
ATGCGCCTTTACAGACGCTGACCCCTTAAACTCCTCTGAGTCGTTAGATTGACTAGCGGATGGTCTGACAACGCGATAAG
GCTGTGCGGTGGACGTTCCATCTCGTTGTGACCGCTACAGAATCACACCCTACATTTCCGCAGCAGGGCTCGCGTGAAAG
TGCGACAGGCTGCCGGGGGGTCCCTCATCACTCACCCGATCTTTGagATTgtAATAAGCCAAACTAGCTCACGTCTAGCC
CTACTACATCTCATGGAGATCTGTGGTCCTCCCAAGATGGGCTCTTATTGAAGCGCTGCGTTGGGATTTCACATGGGGAT
GTCCTTAAAATCATTTATTATGGGAGACGGAGGGATGTAGTCATGATGAGTATCCCCTCGTCGAGAACCATCGGCGTCCC
TCCGCGTCCCCTCAAACGTAACTAGTTGCTTAACGGCGGCATATAAGGCGATCTTTAAACAAACTTAGCCTTATGCACTC
AGTAGCGGCATTTTAGATAG
GAGATGCCACACTAAGTCTGTCAGCCACAAGCCAACATTATGGTTATAGCAACCGAACTCGCATAGCGAACGAGCTTTTA
ATCCAGTCGATGCCGCCTTAAGTAGTAGACGCTCTAATATCAGTCTGAAACCGGACCGGCCAGCACAGGACTGGCCCATA
caCATctGGACGGGAGTGATCGGGTCTATTTAATCGTCAAAATATCTACCTCCTCTCACAATTTTGGCTCATGAGAGGAC
TCTACGCGAGATCGGGCGTGATCTGTCGATGTTACCGTTTGTTCATTTTGTGCAAAGTTCATTTTGACTACACACGTATC
ATCGCGTGAGTTCGCCGAACTACCTGGCGGAGCAGCAGTTGTAAGTCTTTGAGGTTAAGTAAATGCAGTTCCATTGGGGA
GCACCCTTAATGCTAGTGCTCGCAATCTTGCGCCCCCCCACAATGACCACGGATGGTTTTATCTATATTCTGGATTTCTA
GGAATACTGAATCGGAAAGG
TACCAAGTACTTTCAGGACGCTGTGGGTATCAGTGTCATATTACGAGATCACGGCAGTTTACCGAAGAAACAACTGTTTA
AAAACTTACTGTGTGCCTTAaCGGgAttTATCCAACTCGGTGTCACACGTTTCCTGACTTTGCTGGGCCTGTCAGTGCTA
CATGATTTACTATTTACCCAGCAACGATATCCCGCGAGAGGGAGCGCCGCCCAGCACCTCATCACATAAGGGCATCTGAT
TGAAAACCATGGGGGTCGAAAAGTCATTAGTATATAACTGGATTAAACATTCAATTTAGCCAGGTTATTGTGTAATCTTG
TCGGACCCTTCTTAAAACGGCTGAAAAGCGGGGATGGAGGGCCGTGGGCTTTTAGCCCCGCTTGAGCTCAGGGCCCCCCA
GCTACTAAACTTGCCATCATCACTAACGTAGCCCGAGAGTCTGTTTCTGTTCGAGCCTGTATCATGCAAGGGTAATCAAG
TGTTTGACCATTCGAAGAAC
GGTTTATACCTCCCTTTATATCGAGAAATGGGGGTAGACAGGTGACGAACGACCACAGAGCAGACGGCGCGCGTTGGCGA
TCACCGAGAAAACGGGCATCCTAACGAAAACAAGCGCACATCCTCACTGTTCTGTTGGATAAAACCTTGCTACGTTTTGA
GTATTAGTAGGACCGTAGCAATCAGGAAACCGACAGCTCTTTCGGCCCATTATacGtACaACATCCCCAATAAGAACCTA
GGAGTGCTCACAAACTTAGTACATACCCCTAGCGTGTTATAGAGACCTATGAGTGACTCCTTCCCATGACGGTTTTTTAA
GAAGTGGAACCCGTTAGGAGAATTATTCACAACCCGGTGGTGTAGTGAATCTACGTCTATCAGAGGTTATCCTATTAGAA
TACTTCTCGTGCATCTGAGTGACTCTCTGATCTTACGGCCCCTGGTGTCTTGGGCGTTAGAAGCAGTGTCTGTACCCAGA
AAAACCAACTACCCACGAGA
AGATATCGCGCCGACGGGATTATGTAGGCCAAAGGCTCTCACCGCGCATTAGGGTAGTCAATGCACCGGCATCTTGTCTC
AAGGACCAACTAAGGCTAGTTCAATCCTCAATAGCGATCATCCAACGAAGTAGTGCAGCGCATTCGACGATTGCTTATTT
GGGGAGAGGCAGGCACAGTATGCGGTACAGGATAGCTTGGGGGCGATAGCGCCGTTGCTAAACCACCGTCTGTGCACAAC
CCCCGTCAGATAAGCGGTCTATGTAAGGCAGCATTCCACGGCTTGTAGCCGGTAGTACCCCGAGAGGTACAAACCCTCGC
GTTAcAtTTgAaCTGCGGAAACATCTAATAAATAAACTGCTGCTACTCGGCACAATGTCCGCCTTTTCTCTGCACCCGAT
TCTTCACAGACGGCCAAGTCATAGCACTAGCCTCTGACGGGTCTTCGAGTGGCCGGACGCCTAATCGCTCACATCTAGTC
ATTAGAGCTGCAGCGTAAGC
```

Splitting each line in to a different dna text

[6]:
```python
DNA = text.splitlines()
```

[6]:
```
['ATGCGCCTTTACAGACGCTGACCCCTTAAACTCCTCTGAGTCGTTAGATTGACTAGCGGATGGTCTGACAACGCGATA
AGGCTGTGCGGTGGACGTTCCATCTCGTTGTGACCGCTACAGAATCACACCCTACATTTCCGCAGCAGGGCTCGCGTGAA
AGTGCGACAGGCTGCCGGGGGGTCCCTCATCACTCACCCGATCTTTGagATTgtAATAAGCCAAACTAGCTCACGTCTAG
CCCTACTACATCTCATGGAGATCTGTGGTCCTCCCAAGATGGGCTCTTATTGAAGCGCTGCGTTGGGATTTCACATGGGG
ATGTCCTTAAAATCATTTATTATGGGAGACGGAGGGATGTAGTCATGATGAGTATCCCCTCGTCGAGAACCATCGGCGTC
CCTCCGCGTCCCCTCAAACGTAACTAGTTGCTTAACGGCGGCATATAAGGCGATCTTTAAACAAACTTAGCCTTATGCAC
TCAGTAGCGGCATTTTAGATAG',
 'GAGATGCCACACTAAGTCTGTCAGCCACAAGCCAACATTATGGTTATAGCAACCGAACTCGCATAGCGAACGAGCTTT
TAATCCAGTCGATGCCGCCTTAAGTAGTAGACGCTCTAATATCAGTCTGAAACCGGACCGGCCAGCACAGGACTGGCCCA
TAcaCATctGGACGGGAGTGATCGGGTCTATTTAATCGTCAAAATATCTACCTCCTCTCACAATTTTGGCTCATGAGAGG
ACTCTACGCGAGATCGGGCGTGATCTGTCGATGTTACCGTTTGTTCATTTTGTGCAAAGTTCATTTTGACTACACACGTA
TCATCGCGTGAGTTCGCCGAACTACCTGGCGGAGCAGCAGTTGTAAGTCTTTGAGGTTAAGTAAATGCAGTTCCATTGGG
GAGCACCCTTAATGCTAGTGCTCGCAATCTTGCGCCCCCCCACAATGACCACGGATGGTTTTATCTATATTCTGGATTTC
TAGGAATACTGAATCGGAAAGG',
 'TACCAAGTACTTTCAGGACGCTGTGGGTATCAGTGTCATATTACGAGATCACGGCAGTTTACCGAAGAAACAACTGTT
TAAAAACTTACTGTGTGCCTTAaCGGgAttTATCCAACTCGGTGTCACACGTTTCCTGACTTTGCTGGGCCTGTCAGTGC
TACATGATTTACTATTTACCCAGCAACGATATCCCGCGAGAGGGAGCGCCGCCCAGCACCTCATCACATAAGGGCATCTG
ATTGAAAACCATGGGGGTCGAAAAGTCATTAGTATATAACTGGATTAAACATTCAATTTAGCCAGGTTATTGTGTAATCT
TGTCGGACCCTTCTTAAAACGGCTGAAAAGCGGGGATGGAGGGCCGTGGGCTTTTAGCCCCGCTTGAGCTCAGGGCCCCC
CAGCTACTAAACTTGCCATCATCACTAACGTAGCCCGAGAGTCTGTTTCTGTTCGAGCCTGTATCATGCAAGGGTAATCA
AGTGTTTGACCATTCGAAGAAC',
 'GGTTTATACCTCCCTTTATATCGAGAAATGGGGGTAGACAGGTGACGAACGACCACAGAGCAGACGGCGCGCGTTGGC
GATCACCGAGAAAACGGGCATCCTAACGAAAACAAGCGCACATCCTCACTGTTCTGTTGGATAAAACCTTGCTACGTTTT
GAGTATTAGTAGGACCGTAGCAATCAGGAAACCGACAGCTCTTTCGGCCCATTATacGtACaACATCCCCAATAAGAACC
TAGGAGTGCTCACAAACTTAGTACATACCCCTAGCGTGTTATAGAGACCTATGAGTGACTCCTTCCCATGACGGTTTTTT
AAGAAGTGGAACCCGTTAGGAGAATTATTCACAACCCGGTGGTGTAGTGAATCTACGTCTATCAGAGGTTATCCTATTAG
AATACTTCTCGTGCATCTGAGTGACTCTCTGATCTTACGGCCCCTGGTGTCTTGGGCGTTAGAAGCAGTGTCTGTACCCA
GAAAAACCAACTACCCACGAGA',
 'AGATATCGCGCCGACGGGATTATGTAGGCCAAAGGCTCTCACCGCGCATTAGGGTAGTCAATGCACCGGCATCTTGTC
TCAAGGACCAACTAAGGCTAGTTCAATCCTCAATAGCGATCATCCAACGAAGTAGTGCAGCGCATTCGACGATTGCTTAT
TTGGGGAGAGGCAGGCACAGTATGCGGTACAGGATAGCTTGGGGGCGATAGCGCCGTTGCTAAACCACCGTCTGTGCACA
ACCCCCGTCAGATAAGCGGTCTATGTAAGGCAGCATTCCACGGCTTGTAGCCGGTAGTACCCCGAGAGGTACAAACCCTC
GCGTTAcAtTTgAaCTGCGGAAACATCTAATAAATAAACTGCTGCTACTCGGCACAATGTCCGCCTTTTCTCTGCACCCG
ATTCTTCACAGACGGCCAAGTCATAGCACTAGCCTCTGACGGGTCTTCGAGTGGCCGGACGCCTAATCGCTCACATCTAG
TCATTAGAGCTGCAGCGTAAGC']
```

1.0.2 Randomized Motif Search Functions

This function takes random motifs with length 'k' from each line. This function is also used in gibbs sampler

[1073]:
```python
def randomKmer(DNA, k = 10):
    randMotif = [[] for a in range(len(DNA))] #creates a list for motifs
    for i in range(len(DNA)): #iterates thru lines
        startPoint = rd.randint(0,len(DNA[i])-k) #randomly selects starting␣
 ↪index
        for m in range(k):
            randMotif[i].append(list(DNA[i])[startPoint + m]) #adds each letter␣
 ↪until the size is reached
        randMotif[i] = ''.join(randMotif[i])
    return randMotif
print(randomKmer(DNA,20))
```

```
['TCGCATTGGTCTCCGTCGTG', 'TTGCTGCGACCGCGATCATA', 'TTCCTAGTCCCAAACTACTT',
'TCCGCAATATACAGCTATTC', 'TCGGGGGACCACCGACGGGT', 'GCTCTCCGGGACGTCCGGCA',
'ATCCTTCTGGGCATCGTCTC', 'AGCAGCGTTGATCCCCCCTC', 'TGTTCTTACAGTATAATCTT',
'GTATGTTTCCTAGGTATCGT']
```

This function finds out the probabilities of motif lists we give it to them. This is pretty important for us to find out and choose which motifs we'll get from DNA text lines Since we need to iterate through each motif first and character the second, the for loop is inverted from the conventional ones.

```
[494]: def probabilities(randomMotifList):
    score = 0
    motifs = randomMotifList
    motifProb = [{'A':0.0,'T':0.0,'G':0.0,'C':0.0} for i in␣
 ↪range(len(motifs[0]))] #creates a dictionary for each characters probability
    #print(motifs)
    for i in range(len(motifs[0])): #iterates through every character
        A,T,G,C = 0,0,0,0
        for m in range(len(motifs)): #iterates through every motif to add save␣
 ↪their char counts
            if(motifs[m][i].upper() == "A"): A = A + 1
            elif(motifs[m][i].upper() == "T"): T = T + 1 #increase the char␣
 ↪found
            elif(motifs[m][i].upper() == "G"): G = G + 1
            elif(motifs[m][i].upper() == "C"): C = C + 1
            else:
                #return
                print("Unknown Character in motif sequence")
        #print(i,A,T,G,C,A+T+G+C)
        motifProb[i]['A'] = A/(A+T+G+C) #find the probability for each
        motifProb[i]['T'] = T/(A+T+G+C)
        motifProb[i]['G'] = G/(A+T+G+C)
        motifProb[i]['C'] = C/(A+T+G+C)

    consensusMotif = []
    for i in range(len(motifs[0])): #find the consensus motif
        consensusMotif.append(max(motifProb[i], key=motifProb[i].get))

    consensusMotif = ''.join(consensusMotif)

    for a in range(len(motifs)): # find the consensus motifs score
        for i in range(len(consensusMotif)):
            if(consensusMotif[i] != motifs[a][i]): score = score + 1
    #print("Consensus Motif: ", consensusMotif, "Score:", score)
    return motifProb, score
```

```
print(probabilities(randomKmer(DNA,k=5)))
```

```
([{'A': 0.1, 'T': 0.3, 'G': 0.3, 'C': 0.3}, {'A': 0.1, 'T': 0.1, 'G': 0.2, 'C':
0.6}, {'A': 0.3, 'T': 0.2, 'G': 0.2, 'C': 0.3}, {'A': 0.4, 'T': 0.3, 'G': 0.2,
'C': 0.1}, {'A': 0.4, 'T': 0.2, 'G': 0.3, 'C': 0.1}], 32)
```

This function finds out the most probable motifs in for every line and every possible iteration of sliding. Then returns its list of motifs

```
[1171]: def mostProbableMotifs(DNA,probabilityList,k=10):
    lineNum = 0
    newMotifs = ["" for i in range(len(DNA))]
    if(len(probabilityList)!=2):
        probList = probabilityList
    else:
        probList, _ = probabilityList #probabilities(randomKmer(DNA))

    for lineNum in range(len(DNA)): #iterates lines in DNA
        kMer = [{'motif':"", 'prob':1.0000000000000} for i in␣
↪range(len(DNA[lineNum])-k)] #creates a new kMer list for that line
        for m in range(len(DNA[lineNum])-k): #slides k length kMer in that line
            kMer[m]['motif'] = DNA[lineNum][m:m+k] #saves the motif of that k␣
↪length kMer in that line
            for i in range(k): #iteratively multiplies probabilities to achieve␣
↪probability of that motif
                kMer[m]['prob'] = kMer[m]['prob']*probList[i][DNA[lineNum][m+i].
↪upper()]
            #if(kMer[m]['prob']>0):
                #print(kMer[m])

        maxMotif = 0.0000000000000
        for l in range(len(kMer)): #to have the maximum probable motif in that␣
↪line
            if(maxMotif < kMer[l]['prob']):
                maxMotif = kMer[l]['prob']
                newMotifs[lineNum] = kMer[l]['motif'] #update the maximum for␣
↪that line number

        #print("Line:",lineNum,"Motif:",newMotifs[lineNum],"Probability:
↪",maxMotif)

    return newMotifs
print(mostProbableMotifs(DNA,probabilities(randomKmer(DNA))))
```

```
['AGTTTACTCC', 'AATATCCTAA', 'AGAAATCACG', 'AAATTGCTAT', 'AGTACTCTAT',
'GAAGTACTAT', 'AAAACACGCA', 'AGGGCTCTAA', 'AGGATTCTCG', 'GAAACACATA']
```

This was what I originally planned the function would be but after 3 or 5 iterations it reaches a maximum point for that random start and couldn't go further. So I implemented another better version below that uses this.

[1172]:
```
def RandomizedMotifStarter(DNA, k=10, t=10):
    check = 0
    bestMotifs = randomKmer(DNA,k)
    probs, initialScore = probabilities(bestMotifs)
    for i in range(t):
        probMotifs = mostProbableMotifs(DNA,probs,k)
        probs, score = probabilities(probMotifs)
        if(score < initialScore):
            check = 0
            bestMotifs = probMotifs
            initialScore = score
        check = check+1
        if(check >= 50):
            print("I give up after", check, "iterations")
            break

    motifProb , finalScore = probabilities(bestMotifs)

    consensusMotif = []
    for i in range(k): #find the consensus motif
        consensusMotif.append(max(motifProb[i], key=motifProb[i].get))

    consensusMotif = ''.join(consensusMotif)
    #print("Consensus: ", consensusMotif, "Score: ", finalScore)

    return bestMotifs, score, consensusMotif
```

This function calls the earlier one a bunch of times and compares their scores and takes the best one. With this, we reach a much better solution because we compare and converge from different starting points. You can toy with all the parameters for best score value.

[1213]:
```
def RandomizedMotifSearch(DNA, k=10, t=10, randomNum=10): #Random
    print("---------------------------- Random Motif Search Started␣
↪----------------------------")
    bestMotifs, initialScore, consensusMotif = RandomizedMotifStarter(DNA,k,t)
    print("Consensus: ", consensusMotif, "Score: ", initialScore)
    for i in range(randomNum-1):
        probMotifs, score, consensusMotif = RandomizedMotifStarter(DNA,k,t)
        print("Consensus: ", consensusMotif, "Score: ", score)
        if(i%2 == 0):
            print("---------------------------- Random Batch",i+2,␣
↪"----------------------------")
        if(score < initialScore):
            bestMotifs = probMotifs
            initialScore = score


    print("\nConclusion after", randomNum,"random batch iteration:")
    motifProb , finalScore = probabilities(bestMotifs)

    consensusMotif = []
    for i in range(k): #find the consensus motif
        consensusMotif.append(max(motifProb[i], key=motifProb[i].get))

    consensusMotif = ''.join(consensusMotif)
    print("Consensus: ", consensusMotif, "Score: ", finalScore)

    print("Best Motifs: ", bestMotifs)

    return consensusMotif, finalScore
```

1.0.3 Gibbs Sampler Functions

This function is similar to the randomized motif search's function. The difference being that it skips the line we chose and have different approach to calculating probabilities.

[1159]:
```
def probabilitiesGibbs(randomMotifList,lineToIgnore):
    #print(randomMotifList,"length" ,len(randomMotifList))
    #print("line to ignore", lineToIgnore)
    if(lineToIgnore>=len(randomMotifList)):
        return "Error: The line number to ignore is greater than lines␣
↪contained"

    score = 0
    motifs = randomMotifList
    motifProb = [{'A':0.0,'T':0.0,'G':0.0,'C':0.0} for i in␣
↪range(len(motifs[0]))] #creates a dictionary for each characters probability
    #print(motifs)
    for i in range(len(motifs[0])): #iterates through every character
        A,T,G,C = 0,0,0,0
        for m in range(len(motifs)): #iterates through every motif to add save␣
↪their char counts
            if(m==lineToIgnore): continue #skips the chosen line
            if(motifs[m][i].upper() == "A"): A = A + 1
            elif(motifs[m][i].upper() == "T"): T = T + 1 #increase the char␣
↪found
            elif(motifs[m][i].upper() == "G"): G = G + 1
            elif(motifs[m][i].upper() == "C"): C = C + 1
            else:
                #return
                print("Unknown Character in motif sequence")
        #print(i,A,T,G,C,A+T+G+C)
        A,T,G,C = A+1,T+1,G+1,C+1
        motifProb[i]['A'] = A/(A+T+G+C) #find the probability for each
        motifProb[i]['T'] = T/(A+T+G+C)
        motifProb[i]['G'] = G/(A+T+G+C)
        motifProb[i]['C'] = C/(A+T+G+C)


    consensusMotif = []
    for i in range(len(motifs[0])): #find the consensus motif
        consensusMotif.append(max(motifProb[i], key=motifProb[i].get))

    consensusMotif = ''.join(consensusMotif)

    return motifProb, consensusMotif
print(probabilitiesGibbs(randomKmer(DNA,k=10),5))
```

```
([{'A': 0.3076923076923077, 'T': 0.3076923076923077, 'G': 0.3076923076923077,
'C': 0.07692307692307693}, {'A': 0.07692307692307693, 'T': 0.5384615384615384,
'G': 0.15384615384615385, 'C': 0.23076923076923078}, {'A': 0.38461538461538464,
'T': 0.23076923076923078, 'G': 0.15384615384615385, 'C': 0.23076923076923078},
{'A': 0.46153846153846156, 'T': 0.3076923076923077, 'G': 0.07692307692307693,
'C': 0.15384615384615385}, {'A': 0.5384615384615384, 'T': 0.15384615384615385,
'G': 0.15384615384615385, 'C': 0.15384615384615385}, {'A': 0.23076923076923078,
'T': 0.23076923076923078, 'G': 0.3076923076923077, 'C': 0.23076923076923078},
{'A': 0.3076923076923077, 'T': 0.23076923076923078, 'G': 0.23076923076923078,
'C': 0.23076923076923078}, {'A': 0.07692307692307693, 'T': 0.23076923076923078,
'G': 0.3076923076923077, 'C': 0.38461538461538464}, {'A': 0.23076923076923078,
'T': 0.3076923076923077, 'G': 0.23076923076923078, 'C': 0.23076923076923078},
{'A': 0.15384615384615385, 'T': 0.23076923076923078, 'G': 0.38461538461538464,
'C': 0.23076923076923078}], 'ATAAAGACTG')
```

This function applies the probabilities we calculated earlier to the skipped line. After that, it adds them up to reach 'C'. Keep in mind this C is not citozine but the denumerator for our formula as given in the study material

```
[1174]: def mostProbableMotifGibbs(DNA,probabilityList,ignoredLine,k=10):
    lineNum = ignoredLine

    if(len(probabilityList)!=2):
        probList = probabilityList
    else:
        probList, _ = probabilityList

    kMer = [{'motif':"", 'prob':1.0000000000000} for i in␣
 ↪range(len(DNA[lineNum])-k)] #creates a new kMer list for thatline

    #print("probList length", len(probList))
    for m in range(len(DNA[lineNum])-k): #slides k length kMer in that line
        kMer[m]['motif'] = DNA[lineNum][m:m+k] #saves the motif of that k␣
↪length kMer in that line
        for i in range(k): #iteratively multiplies probabilities to achieve␣
↪probability of that motif
            kMer[m]['prob'] = kMer[m]['prob']*probList[i][DNA[lineNum][m+i].
↪upper()]
        #if(kMer[m]['prob']>0):
            #print(kMer[m])

    motifList = [] #hold the motif strings of the porbability values
    weights = [] #holds the probability values of motifs
    C = 0.00000000000000000000000001 #to handle division by zero
    for i in range(len(kMer)):
        C = C + kMer[i]['prob']

    for i in range(len(kMer)): #create the distribution
        motifList.append(kMer[i]['motif'])
        weights.append(kMer[i]['prob']/C)

    chosenMotif = rd.choices(motifList, weights, k = 1) #choose from␣
↪distribution

    return chosenMotif
print(mostProbableMotifGibbs(DNA,probabilitiesGibbs(randomKmer(DNA),5),5))
```

```
['ACGTCTCTCT']
```

After choosing a better motif for the skipped line, we insert it to our original motif list. By doing that we completed everything necessary for one iteration.

This function also computes the score. Because we needed the updated version of the skipped line to compute it.

[855]:
```
def insertTheIgnored(motifList, chosenMotif, ignoreIndex,consensusMotif):
    score = 0
    newMotifList = ["" for i in range(len(motifList))]

    for i in range(len(motifList)):
        if(i == ignoreIndex):
            newMotifList[i] = ''.join(chosenMotif)
        else:
            newMotifList[i] = motifList[i]

    for a in range(len(newMotifList)): # find the consensus motifs score
        #print(newMotifList[a])
        for i in range(len(consensusMotif)):
            pass
            if(consensusMotif[i] != newMotifList[a][i]): score = score + 1

    return newMotifList, score
```

This is the main iterator for our algorithm. This somewhat closer to what you'll find in the text book pseudo-codes. Since this is only from one starting point, the score is not so great

```
[1175]: def GibbsMotifStart(DNA, k=10, t=10):
    check = 0
    initialMotifs = randomKmer(DNA,k)
    indexToIgnore = rd.randint(0,len(DNA)-1)
    probList = probabilitiesGibbs(initialMotifs,indexToIgnore)[0]
    consensus = probabilitiesGibbs(initialMotifs,indexToIgnore)[1]
    chosenMotif = mostProbableMotifGibbs(DNA,probList,indexToIgnore,k)
    updatedMotifList, initialScore =␣
 ↪insertTheIgnored(initialMotifs,chosenMotif,indexToIgnore,consensus)
    for i in range(t):
        indexToIgnore = rd.randint(0,len(DNA)-1)
        if(len(probabilitiesGibbs(updatedMotifList,indexToIgnore))>2):
            print(probabilitiesGibbs(updatedMotifList,indexToIgnore))

        probList, consensus = probabilitiesGibbs(updatedMotifList,indexToIgnore)
        chosenMotif = mostProbableMotifGibbs(DNA,probList,indexToIgnore,k)
        updatedMotifList, score =␣
 ↪insertTheIgnored(updatedMotifList,chosenMotif,indexToIgnore,consensus)
        if(score<initialScore):
            check = 0
            initialScore = score
            bestMotifs = updatedMotifList
        check = check+1
        if(check >= 50):
            print("I give up after", check, "iterations")
            break


    motifProb , finalScore = probabilities(bestMotifs)

    consensusMotif = []
    for i in range(k): #find the consensus motif
        consensusMotif.append(max(motifProb[i], key=motifProb[i].get))

    consensusMotif = ''.join(consensusMotif)
    #print("Consensus: ", consensusMotif, "Score: ", finalScore)
    return bestMotifs, finalScore, consensusMotif
GibbsMotifStart(DNA)
```

```
[1175]: (['GCTAATCACG',
          'ATAGATACCT',
          'GAATGTAAAG',
          'CGAGTTTAAG',
          'GCATATTCTG',
          'ACAGGCTCCG',
          'TGAGAAAAAA',
          'CGCACATCGT',
          'ACAGACTCAG',
          'TCAGAAACAC'],
         45,
         'ACAGATTCAG')
```

That's why I also implemented a somekind of initiator. This function randomly initiates and takes the best score from each of the randomly initiated batches. This had a huge impact for the score

```
[1210]: def GibbsSamplerSearch(DNA, k=10, t=10, randomNum=10): #Random
            print("---------------------------- Gibbs Sampler Search Started␣
         ↪----------------------------")
            bestMotifs, initialScore, consensusMotif = GibbsMotifStart(DNA,k,t)
            print("Consensus: ", consensusMotif, "Score: ", initialScore)
            for i in range(randomNum-1):
                probMotifs, score, consensusMotif = GibbsMotifStart(DNA,k,t)
                print("Consensus: ", consensusMotif, "Score: ", score)

                if(i%2 == 0):
                    print("---------------------------- Random Batch",i+2,␣
         ↪"----------------------------")
                if(score < initialScore):
                    bestMotifs = probMotifs
                    initialScore = score

            print("\nConclusion after", randomNum,"random batch iteration:")
            motifProb , finalScore = probabilities(bestMotifs)

            consensusMotif = []
            for i in range(k): #find the consensus motif
                consensusMotif.append(max(motifProb[i], key=motifProb[i].get))

            consensusMotif = ''.join(consensusMotif)
            print("Consensus: ", consensusMotif, "Score: ", finalScore)

            print("Best Motifs: ", bestMotifs)
            return consensusMotif, finalScore
```

1.0.4 Algorithm Testing and Comparison

The final tests with the actual parameters presented in assignment with all the functions.

First we generate and read a DNA text file.

Then we split the text file we read line by line and contain it in a list

Then we initiate our general parameters for our functions

[1089]:
```
f = open("Generated.txt", "w")
f.write(generateText(length=10, mutation=4, lineLength=500, numberOfLines=10))
f.close()
f = open("Generated.txt", "r")
text = f.read()
f.close()
#print(text)
DNA = text.splitlines()
t = 200 #if it can't converge after 200 just take it
randomNum = 10 #how many random starting points?
```

[1214]:
```
consensus_rand9, score_rand9 =␣
↪RandomizedMotifSearch(DNA,k=9,t=t,randomNum=randomNum)
```

```
---------------------------- Random Motif Search Started
----------------------------
I give up after 50 iterations
Consensus:  GCAACGTCC Score:  23
I give up after 50 iterations
Consensus:  TAAGGGGAT Score:  29
---------------------------- Random Batch 2 ----------------------------
I give up after 50 iterations
Consensus:  ATAGTAGGG Score:  29
I give up after 50 iterations
Consensus:  TCTACGGCT Score:  25
---------------------------- Random Batch 4 ----------------------------
I give up after 50 iterations
Consensus:  TTCTACTGA Score:  32
I give up after 50 iterations
Consensus:  GCCGTATCG Score:  26
---------------------------- Random Batch 6 ----------------------------
I give up after 50 iterations
Consensus:  TAGCATCTT Score:  32
I give up after 50 iterations
Consensus:  TAGCACGAA Score:  30
---------------------------- Random Batch 8 ----------------------------
I give up after 50 iterations
Consensus:  AGGGGAGGC Score:  24
I give up after 50 iterations
Consensus:  GCGGAGGGC Score:  26
---------------------------- Random Batch 10 ----------------------------

Conclusion after 10 random batch iteration:
```

```
Consensus:  GCAACGTCC Score:  23
Best Motifs:  ['GCATCGTCC', 'GAAACCCCC', 'GCATGGAGC', 'GCAACGTCA', 'GGATCCCCC',
'GCAACGAGC', 'GCACCGTCC', 'GCAGCGTCC', 'GCAACGCCC', 'GAATCCGGC']
```

[1215]:
```
consensus_rand10, score_rand10 =␣
↪RandomizedMotifSearch(DNA,k=10,t=t,randomNum=randomNum)
```

```
---------------------------- Random Motif Search Started
----------------------------
I give up after 50 iterations
Consensus:  TACCTGGAGG Score:  35
I give up after 50 iterations
Consensus:  TAGTAGATGC Score:  40
---------------------------- Random Batch 2 ----------------------------
I give up after 50 iterations
Consensus:  AGAACTACAT Score:  30
I give up after 50 iterations
Consensus:  ATAGCTTGAG Score:  33
---------------------------- Random Batch 4 ----------------------------
I give up after 50 iterations
Consensus:  TTATTATTAT Score:  38
I give up after 50 iterations
Consensus:  CGGAGCGACC Score:  33
---------------------------- Random Batch 6 ----------------------------
I give up after 50 iterations
Consensus:  GTTACTGAGA Score:  33
I give up after 50 iterations
Consensus:  CTTGCTCGCG Score:  33
---------------------------- Random Batch 8 ----------------------------
I give up after 50 iterations
Consensus:  AGTACAAAGC Score:  33
I give up after 50 iterations
Consensus:  CAATACATAT Score:  34
---------------------------- Random Batch 10 ----------------------------

Conclusion after 10 random batch iteration:
Consensus:  AGAACTACAT Score:  30
Best Motifs:  ['AGAACAACAT', 'ATGAATACCT', 'ATAACTACAC', 'AGGACAAGGT',
'GGGACTCAAG', 'AGATATACGT', 'AGAACTATAt', 'GGGAGTACGT', 'GGAACTACGG',
'AGAAACACAT']
```

[1216]:
```
consensus_rand11, score_rand11 =␣
↪RandomizedMotifSearch(DNA,k=11,t=t,randomNum=randomNum)
```

```
---------------------------- Random Motif Search Started
----------------------------
I give up after 50 iterations
Consensus:  TAGCGCGAATT Score:  39
```

```
I give up after 50 iterations
Consensus:  TGAAACATTTG Score:  34
---------------------------- Random Batch 2 ----------------------------
I give up after 50 iterations
Consensus:  CTCTCTATCTG Score:  43
I give up after 50 iterations
Consensus:  CTTTCGGAGAT Score:  42
---------------------------- Random Batch 4 ----------------------------
I give up after 50 iterations
Consensus:  ATAGCGTAGTA Score:  45
I give up after 50 iterations
Consensus:  GTCGTGTCGAG Score:  37
---------------------------- Random Batch 6 ----------------------------
I give up after 50 iterations
Consensus:  CACGTCCGGCC Score:  38
I give up after 50 iterations
Consensus:  CCGCGGAGGCT Score:  36
---------------------------- Random Batch 8 ----------------------------
I give up after 50 iterations
Consensus:  AATCCGGAGAA Score:  42
I give up after 50 iterations
Consensus:  GAGAAGCAGCT Score:  38
---------------------------- Random Batch 10 ----------------------------

Conclusion after 10 random batch iteration:
Consensus:  TGAAACATTTG Score:  34
Best Motifs:  ['TGGAACATTTg', 'AGATACCTGTC', 'TGATACAGTCG', 'TGATCCAGTTG',
'TGAAGGATTTT', 'AGATATACGTG', 'GGAAACCTTGA', 'TAaATCATTTG', 'TTAAACTTTGC',
'TGAACCATGTG']
```

[1217]:
```
consensus_gibbs9, score_gibbs9 =␣
↪GibbsSamplerSearch(DNA,k=9,t=t,randomNum=randomNum)
```

```
---------------------------- Gibbs Sampler Search Started
----------------------------
I give up after 50 iterations
Consensus:  GTATGAATG Score:  24
I give up after 50 iterations
Consensus:  CTTTTGCAA Score:  29
---------------------------- Random Batch 2 ----------------------------
I give up after 50 iterations
Consensus:  CTCCGGTCT Score:  30
I give up after 50 iterations
Consensus:  TTAGATATG Score:  33
---------------------------- Random Batch 4 ----------------------------
I give up after 50 iterations
Consensus:  GTGTTAAGC Score:  29
I give up after 50 iterations
```

```
Consensus:  CGTCATTCT Score:  33
---------------------------- Random Batch 6 ----------------------------
I give up after 50 iterations
Consensus:  ATCCCCAAA Score:  31
I give up after 50 iterations
Consensus:  TTTCAGTCG Score:  32
---------------------------- Random Batch 8 ----------------------------
I give up after 50 iterations
Consensus:  AGAATGACC Score:  29
I give up after 50 iterations
Consensus:  CCAGGAGTT Score:  29
---------------------------- Random Batch 10 ----------------------------

Conclusion after 10 random batch iteration:
Consensus:  GTATGAATG Score:  24
Best Motifs:  ['ATATTAAAG', 'GAATGAATG', 'GTATGCATG', 'GAAAAAAGG', 'GTATTAATG',
'GTATGAACG', 'CTAGCAAAT', 'GAATGAATT', 'GTTTAAACT', 'GTAAGAAAG']
```

[1218]:
```
consensus_gibbs10, score_gibbs10 =␣
↪GibbsSamplerSearch(DNA,k=10,t=t,randomNum=randomNum)
```

```
---------------------------- Gibbs Sampler Search Started
----------------------------
I give up after 50 iterations
Consensus:  GCCTTGTAGA Score:  35
I give up after 50 iterations
Consensus:  GGAGCACCAA Score:  32
---------------------------- Random Batch 2 ----------------------------
I give up after 50 iterations
Consensus:  TCGCAGTGCA Score:  39
I give up after 50 iterations
Consensus:  AGCACGTATT Score:  28
---------------------------- Random Batch 4 ----------------------------
I give up after 50 iterations
Consensus:  AATATCTTAG Score:  35
I give up after 50 iterations
Consensus:  TATACGAGTC Score:  40
---------------------------- Random Batch 6 ----------------------------
I give up after 50 iterations
Consensus:  GATAACTTGA Score:  30
I give up after 50 iterations
Consensus:  CAATAACATG Score:  43
---------------------------- Random Batch 8 ----------------------------
I give up after 50 iterations
Consensus:  TGATTGAATC Score:  40
I give up after 50 iterations
Consensus:  GTTTTGAATA Score:  38
---------------------------- Random Batch 10 ----------------------------
```

```
Conclusion after 10 random batch iteration:
Consensus:  AGCACGTATT Score:  28
Best Motifs:  ['ATCACGAGTT', 'CGCACATATG', 'TGCACGGATA', 'AGGAAGTATT',
'GGCATGTATT', 'AGCAGTTGTG', 'AGCAAATATT', 'AGTACGTATA', 'AACGAGGATT',
'AGGACGCACT']
```

[1219]:
```
consensus_gibbs11, score_gibbs11 =␣
↪GibbsSamplerSearch(DNA,k=12,t=t,randomNum=randomNum)
```

```
---------------------------- Gibbs Sampler Search Started
----------------------------
I give up after 50 iterations
Consensus:  CAGGGCATGCAA Score:  42
I give up after 50 iterations
Consensus:  TCCTGCATAGTT Score:  39
---------------------------- Random Batch 2 ----------------------------
I give up after 50 iterations
Consensus:  TTTCTTAAATAT Score:  43
I give up after 50 iterations
Consensus:  AAGGGTTGTGCT Score:  42
---------------------------- Random Batch 4 ----------------------------
I give up after 50 iterations
Consensus:  CCTTGAGGCGAG Score:  47
I give up after 50 iterations
Consensus:  AAGCGTAGCGCG Score:  37
---------------------------- Random Batch 6 ----------------------------
I give up after 50 iterations
Consensus:  CGAACTTATCTT Score:  43
I give up after 50 iterations
Consensus:  TACAATCCATGT Score:  46
---------------------------- Random Batch 8 ----------------------------
I give up after 50 iterations
Consensus:  TGTTAAGTCTTG Score:  38
I give up after 50 iterations
Consensus:  GATTAAATGTTA Score:  51
---------------------------- Random Batch 10 ----------------------------

Conclusion after 10 random batch iteration:
Consensus:  AAGCGTAGCGCG Score:  37
Best Motifs:  ['GAGCTTAGCACG', 'AAGCGCACCGGT', 'AAGCGTTGTGGG', 'GAGCCTAGCGCG',
'ATGCATAATGCT', 'CTGCGTAGGGCG', 'CAGAGTAGCGCG', 'GAGGGTAACCTT', 'AAGCATAGCCTG',
'CAGCGGAGGCTT']
```

[1220]:
```
print("Random Motif Search")
print("k = 9: ")
print("Consensus: ", consensus_rand9, "Score: ", score_rand9)
print("k = 10: ")
print("Consensus: ", consensus_rand10, "Score: ", score_rand10)
print("k = 11: ")
print("Consensus: ", consensus_rand11, "Score: ", score_rand11)

print("Gibbs Sampler Search")
print("k = 9: ")
print("Consensus: ", consensus_gibbs9, "Score: ", score_gibbs9)
print("k = 10: ")
print("Consensus: ", consensus_gibbs10, "Score: ", score_gibbs10)
print("k = 11: ")
print("Consensus: ", consensus_gibbs11, "Score: ", score_gibbs11)
```

```
Random Motif Search
k = 9:
Consensus:  GCAACGTCC Score:  23
k = 10:
Consensus:  AGAACTACAT Score:  30
k = 11:
Consensus:  TGAAACATTTG Score:  34
Gibbs Sampler Search
k = 9:
Consensus:  GTATGAATG Score:  24
k = 10:
Consensus:  AGCACGTATT Score:  28
k = 11:
Consensus:  AAGCGTAGCGCG Score:  37
```

1.1 Conclusion:

As you can see I implemented a random motif search algorithm and gibbs sampler from scratch and improved it as best as I could. Gibbs was especially challanging with all its list index management issues.

Iterating from same random starting point was pretty much pointless most of the time. Because even the 11 k-mer motif converged almost immidiately. Starting from different random points really improved the convergence point and should be considered more for every random initating algorithm.

I tried to compute time but they were quick enough to baffle my computers timer. So I did not used it in the final product. Yet in general, Randomized Motif Search is slower than Gibbs Sampler. I would say Gibbs is almost two times faster than Randomized Motif Search. Given that, Randomized Motif Search is slightly better at converging as you can see from the scores.

In conclusion, with very large data sets, Gibbs would be more usefull because of its speed. Even though it slightly scores less, it can iterate much more and possible get better results in the same time.

Emre Erdem - 150115501