Motif Discovery: Randomised Motif Search

Raja Pavan Karthik
Department of Computer Science
and Engineering
Amrita Vishwa Vidyapeetham
Amritapuri, Kerala, India
amenu4aie20060@am.students.amrita.edu

Abstract **—** Motifs are crucial patterns in biological sequences that have numerous applications. Motif search is an important step in obtaining meaningful patterns from biological data. However, most of the existing algorithms are deterministic and the role of randomisation in this area is still unexploited. Randomized Motif Search, Two kinds of algorithms for solving this problem can be found in the literature, namely, exact and approximate. We introduce a third kind, namely, randomized algorithms with an estimate on the probability of identifying the random motif. We present one such algorithm. An experimental study of this algorithm has yielded promising results.

KEYWORDS

Motif, Random motif search, K-mer, Algorithm, Profile, Dna, String

INTRODUCTION

A randomized algorithm is an algorithm that employs a degree of randomness as part of its logic.
Making random algorithmic decisions may sound like a disastrous idea, just imagine a chess game in which every move would be decided by rolling a die.The main advantage is that no input can reliably produce worst-case results because the algorithm runs differently each time. These algorithms are commonly used in situations where no exact and fast algorithm is known. Since a single run of RANDOMIZED MOTIF SEARCH may generate a rather poor set of motifs, bioinformaticians usually run this algorithm thousands of times.

For Motif Finding, we use a sub-category of randomized algorithms called Monte Carlo Algorithm. We can run it many times and choose the best output. On each run, they begin from a new randomly selected set of k-mers, selecting the best set of k-mers found in all these runs. They are not guaranteed to return an exact solution. They quickly find approximate solutions. Las Vegas algorithms are the subset of Monte Carlo algorithms that always produce the correct answer. Because they make random choices as part of their work, the time taken might vary between runs even with the same input.

RANDOMISED MOTIF SEARCH

What is Motif?

Motif is a region (a subsequence) of protein or DNA sequence that has a specific structure. Motifs are candidates for functionally important sites. Presence of a motif may be used as a base of protein classification.
Biological sequence motifs are defined as short, usually fixed length, sequence patterns that may represent important structural or functional features in nucleic acid and protein sequences such as transcription binding sites, splice junctions, active sites, or interaction interfaces.

They occur in an exact or approximate form within a family or a subfamily of sequences. Motif discovery is therefore an important challenge in bioinformatics and numerous methods have been developed for the identification of motifs shared by a set of functionally related sequences.

Randomized algorithms may be non intuitive because they lack the control of traditional algorithms. Some randomized algorithms are Las Vegas algorithms, which deliver solutions that are guaranteed to be exact, despite the fact that they rely on making random decisions. Yet most randomized algorithms, including the motif finding algorithms that we will consider in this chapter, are Monte Carlo algorithms. These algorithms are not guaranteed to return exact solutions, but they do quickly find approximate solutions. Because of their speed, they can be run many times, allowing us to choose the best approximation from thousands of runs.

We previously defined PROFILE(Motifs) as the profile matrix constructed from a collection of k-mers Motifs in Dna. Now, given a collection of strings Dna and an arbitrary 4 ⇥ k matrix Profile, we define MOTIFS(Profile, Dna) as the collection of k-mers formed by the Profile-most probable k-mers in each sequence from Dna.
Why would we do this? Because our hope is that MOTIFS(PROFILE(Motifs), Dna) has a better score than the original collection of k-mers Motifs. We can then form the profile matrix of these k-mers, and use it to form the most probable k-mers.

We can continue to iterate..... PROFILE(MOTIFS(PROFILE(MOTIFS(PROFILE(Motifs), Dna)), Dna)).. for as long as the score of the constructed motifs keeps improving, which is exactly what RANDOMIZED MOTIF SEARCH does. To implement this algorithm, you will need to randomly select the initial collection of k-mers that form the motif matrix Motifs. To do so, you will need a random number generator (denoted RANDOM(N)) that is equally likely to return any integer from 1 to N. You might like to think about this random number generator as an unbiased N-sided die.

Since a single run of RANDOMIZED MOTIF SEARCH may generate a rather poor set of motifs, bioinformaticians usually run this algorithm thousands of times. On each run, they begin from a new randomly selected set of k-mers, selecting the best set of k-mers found in all these runs.

WORKING

At first glance, RANDOMIZED MOTIF SEARCH appears to be doomed. How can this algorithm, which starts from a random guess, possibly finds anything useful? To explore RANDOMIZEDMOTIFSEARCH, let's run it on five short strings with the implanted (4, 1)-motif ACGT (shown in upper case letters below) and imagine that it chooses the following 4-mers Motifs (shown in red) at the first iteration. As expected, it misses the implanted motif in nearly every string.

We can now compute the probabilities of every 4-mer in Dna based on this profile matrix. We select the most probable 4-mer in each row above as our new collection Motifs For the Subtle Motif Problem with implanted 15-mer AAAAAAAAGGGGGGG, when we run RANDOMIZED MOTIF SEARCH 100,000 times (each time with new randomly selected k-mers), it returns the 15-mers as the lowest scoring collection.

Motifs across all iterations, resulting in the consensus string AAAAAAAAacaGGGG with score 43. These strings are only slightly less conserved than the collection of implanted (15,

4)-motifs with score 40 (or the motif returned by GREEDY MOTIF SEARCH with score 41), and it largely captures the implanted motif. Furthermore, unlike GREEDY MOTIF SEARCH, RANDOMIZEDMOTIFSEARCH can be run for a larger number of iterations to discover better and better motifs.

Although the motifs returned by RANDOMIZEDMOTIFSEARCH are slightly less conserved than the motifs returned by MEDIANSTRING, RANDOMIZEDMOTIFSEARCH has the advantage of being able to find longer motifs (since MEDIANSTRING becomes too slow for longer motifs). In the epilogue, we will see that this feature is important in practice.

METHODS

For example, consider the following Profile and Dna:

```
Profile                Dna
                       ttaccttaac
A: 4/5  0   0  1/5     gatgtctgtc
C:  0  3/5 1/5  0      acggcgttag
G: 1/5 1/5 4/5  0      ccctaacgag
T:  0  1/5  0  4/5     cgtcagaggt
```

Taking the Profile-most probable 4-mer from each row of Dna produces the following 4-mers (shown in red):

```
                       ttaccttaac
                       gatgtctgtc
MOTIFS(Profile, Dna)   acggcgttag
                       ccctaacgag
                       cgtcagaggt
```

- ➔ Begin from a collection of randomly chosen k-mers Motifs in Dna
- ➔ Construct PROFILE(Motifs)
- ➔ Use this profile to generate a new collection of k-mers: MOTIFS(PROFILE(Motifs),Dna)
- ➔ Then form the profile matrix of these k-mers, PROFILE(MOTIFS(PROFILE(Motifs), Dna)) ,
- ➔ Use it to form the most probable k-mer Motifs: MOTIFS(PROFILE(MOTIFS(PROFILE(Motifs), Dna)),
- ➔ We can continue to iterate . . . PROFILE(MOTIFS(PROFILE(MOTIFS(PROFILE(Motifs), Dna)), Dna))..

```
##ALGORITHM
RANDOMIZEDMOTIFSEARCH(Dna, k, t)
randomly select k-mers Motifs =
(Motif1, ... , Motift) in each
string from Dna
BestMotifs Motifs
while forever
Profile PROFILE(Motifs)
Motifs MOTIFS(Profile, Dna)
if SCORE(Motifs) < SCORE(BestMotifs)
BestMotifs Motifs
else
return BestMotifs
```

RESULTS

```
ACTGGAGGGCACCTA
TCTTACGTACAGGTA
TCAACAGTACAGGTA
TCTCACTTACAGGTA
TCTCTAGTATGAGTA
TCTCTAGTTAGGGTA
ACTCTAGTACAGGCT
TCTCTAGTACGCTTA
TCTCTAGTACAATGA
TCTCCTCTACAGGTA
TCTCTAACGCAGGTA
```

```
ATCCTAGTACAGGTA
TTATTAGTACAGGTA
TCTCTGTAACAGGTA
TCTCTAGTACAGCGT
TCTCTGTGACAGGTA
ATTCTAGTACAGGTG
TCTCTAGGCGAGGTA
TCTACGGTACAGGTA
TCTCTACGGCAGGTA
```

CONCLUSION

We can run RANDOMIZED MOTIF SEARCH many times, so that it will almost certainly catch at least one implanted motif. Unfortunately, capturing a single implanted motif is often insufficient to steer RANDOMIZED MOTIF SEARCH to an optimal solution. Therefore, since the number of starting positions of kmers is huge, the strategy of randomly selecting motifs is often not as successful as in the simple example above. The chance that these randomly selected k-mers will be able to guide us to the optimal solution is relatively small.Although the probability that randomly selected kmers match all implanted motifs is negligible, the probability that they capture at least one implanted motif is significant.

REFERENCES

[1] Compeau, Phillip. Bioinformatics

Algorithms. Active Learning Publishers, 2016.

[2] "Implement RandomizedMotifSearch Solved

by 492." ROSALIND,

rosalind.info/problems/ba2f/.

[3] "Bioalgorithms.info." Home,

www.bioalgorithms.info/.

```
import numpy as np

def all_kmers(seq, k):
    return [seq[i:i+k] for i in range(len(seq) - k + 1)]

## return the k-mer composition of a string

def kmer_to_matrix(kmer):
    d = {'A':[1, 0, 0, 0], 'C':[0, 1, 0, 0], 'G':[0, 0, 1, 0], 'T':[0, 0, 0, 1]}
    return np.array([d[i] for i in kmer]).T

## Convert a single k-mer to a (4, k) matrix in the order of 'ACGT'.

def matrix_to_kmer(m):
    d = {(1, 0, 0, 0): 'A', (0, 1, 0, 0): 'C', (0, 0, 1, 0): 'G', (0, 0, 0, 1):
'T'}
    return ''.join([ d[tuple(i)] for i in m.T ])

## Convert a (4, k) k-mer matrix in the order of 'ACGT' back to DNA sequence.

def randomized_motifsearch(dna, k, t, iteration):
    dna_m = np.array([[kmer_to_matrix(j) for j in all_kmers(i, k)] for i in dna])

## converting all kmers in dna into a matrix dna_m;
## the shape of the matrix is (t, len(seq)-k+1, 4, k)

    score_best = 0
    for r in range(iteration):
        #random draw from each row in dna
        draw = np.random.randint(0, high=len(dna[0])-k+1, size=t)
        motifs = dna_m[range(t), draw]
        while True:

##compute a probability profile based on the k-mers randomly drawn

            profile = (motifs.sum(0) + 1) / (t + 4)

##select a new set of k-mers best-matching the profile

            motifs = dna_m[range(t),np.product((dna_m * profile).sum(2),
axis=2).argmax(1)]
```

```
            score_motifs = motifs.sum(0).max(0).sum()
            if score_motifs > score_best:
                best_motifs = motifs
                score_best = score_motifs
            else:
                print(r, score_best, score_motifs)
                break
    return [matrix_to_kmer(i) for i in best_motifs]
f = open('rosalind_ba2f.txt').readlines()
k, t = [int(i) for i in f[0].rstrip().split()]
dna = [i.strip() for i in f[1:]]

result = '\n'.join(randomized_motifsearch(dna, k, t, 1000))
print(result)
open('rosalind_ba2f_sub.txt', 'wt').write(result)
```