CS-E4002 Special Course in Computer Science: Genetic Algorithms. Summer 2017
Advisor: prof. Sami Khuri, San Jose State University, CA, USA

A SIMPLE GENETIC ALGORITHM APPROACH TO THE CLOSEST STRING PROBLEM IN BIOINFORMATICS

Anna Pawelczyk[1]

Abstract. Having a set of n strings $S = \{s_1, s_2, ..., s_n\}$ each of length L and over alphabet A, the Closest String Problem (CSP) searches for a string s_{sol} that is within the smallest Hamming distance d to each $s_i \epsilon S$ [1]. This problem is NP-complete [3].

CSP has been extensively studied in computational biology and coding theory[2]. One of the targets is to design a new DNA or amino acids sequence that is very similar to each given sequence. The sample applications are PCR prime design, genetic probe design, motif finding and antisense drug design [2].

In this paper we present how the Simple Genetic Algorithm with use of schemata performs in finding the Closest String. We focus on its application in Bioinformatics: finding the closest DNA sequence or finding the closest amino acids sequence. We analyse how the results vary depending on chosen techniques and variables.

Keywords: Closest String Problem, Genetic Algorithm, DNA pattern discovery, amino acids pattern discovery

[1] Poland

Introduction

Given a set of n strings $S = \{s_1, s_2, ..., s_n\}$ each of length L and over alphabet A, the objective for the Closest String Problem (CSP for short) is to find a string s_{sol} of length L and over alphabet A, such that it has a minimum Hamming distance $d(s_{sol}, s_i)$ for every $s_i \epsilon S$.

The problem is known to be NP-complete [3]. Researchers have developed approximation algorithms, fixed-parameter algorithms, use polynomial-time approximation scheme (PTAS) Parameterized algorithms have been then developed to solve the problem when d is small [3].

In this paper we focus on finding a solution to CSP by the application of the Simple Genetic Algorithm with use of schemata. We focus on its application in Bioinformatics which results in choosing specific alphabets (the 4-items DNA alphabet and the 20-items amino acids alphabet) and using sample data from Bioinformatic database GenBank [9].

1. Notation

1.1. Problem instance

A set of n strings $S = \{s_1, s_2, ..., s_n\}$ each of length $|s_i| = L$ and over alphabet A.

1.2. Feasible solution

A string s_{sol} of length L and over alphabet A

1.3. Objective function

A Hamming distance $d(s_{sol}, s_i)$ which is the number of positions at which the corresponding symbols are different [11].

1.4. Optimal solution

s_{sol} where for a Hamming distance $\mathrm{d}(\mathrm{s}_{sol}, s_i)$, the $\max_{s_i \epsilon S} d(s_{sol}, s_i)$ is minimal.

2. Bioinformatics application

Current science puts a lot of effort to investigate the nucleotide sequences in DNA. There are international projects such as databases eg. DNA DataBank of Japan (DDBJ), the European Nucleotide Archive (ENA), and GenBank at NCBI) [9], alignment search tools (eg. Blast [6], BLAT [7], numerous libraries (eg. Biopython [5], Bio++ [4], EMBOSS [8]) etc. As already mentioned, one of the targets is to design a new DNA or amino acids sequence that is very similar to each given sequence. This is an instance of the Closest String Problem.

2.1. Notation - problem instance

Having a set of n sequences $S = \{s_1, s_2, ..., s_n\} : |s_i| = L$, a sequence can be a DNA sequence or amino acids sequence.

A DNA sequence is a sequence of characters over the alphabet
$A = \{A, C, G, T\}, |A| = 4$.
A amino acids sequence is a sequence of characters over the alphabet
$A = \{A, B, C, D, E, F, G, H, I, K, L, M, N, O, P, Q, R, S, T, U, V, W, Y, Z\}, |A| = 20$

We aim at finding a sequence that in the Hammington distance context is the closest to the ones in S. The Hammington distance is here the number of nucleotide differences between two genetic sequences [**Pilcher**].

3. Simple Genetic Algorithm

As CSP is an NP-hard problem, we decided to search for the closest sting with a genetic algorithm. It's representation is the following:

We are given a set of nucleotides (or amino acids). Those are strings of the same length over alphabet of nucleotides (or amino acids). We randomly generate a population which are strings of the same length and alphabet. For an arbitrary number of times we perform fitness estimation for each string from the population. Then, according to the fitness we choose the fittest and with some probability we mate them. Then with a very small probability, we mutate one character in sequence. Those strings will now represent the new population.

For the more detiled description, see [10].

3.1. Fitness function

For a given string s, the fitness function is:

$$F_s = \forall_{s_i \epsilon S} \sum_{k=1}^{L} max(d(s[k], s_i[k])) \tag{1}$$

where $s[k]$ is the k'th character in the string s and the Hamming distance $d(a, b)$ is

$$d(a,b) = \begin{cases} 1 & \text{if a != b} \\ 0 & \text{if a = b} \end{cases} \tag{2}$$

This is the minimization problem. In order to perform a roulette random choice, we need to count for every element of the population an appropriate relative fitness value. It as done as following:

$$F_{irel} = \frac{\frac{1}{F_i}}{\sum_{k=1}^{L} \frac{1}{F_k}} \tag{3}$$

The solution is a string s_{sol}, where each character corresponds either to a nucleotide or amino acid. To find it, we generate randomly

For the details of the Simple Genetic Algorithm implementation, please see [10]

3.2. Schema

As the DNA (or amino acids) sequences often can be very similar, we introduce schema to the algorithm.
For each position $k \epsilon \{1..L\}$ we count over all $s_i \epsilon S$ the percentage of times when a particular char appears at this position. If the percentage is over an appropriate rate (eg. 75%), we mark this char as fixed at this position.
When generating the initial population, at positions marked as fixed, we put the fixed characters.

3.3. Complexity

The time complexity of this implementation of the Simple Genetic Algorithm is $O(Generations * |Population| * L * n)$.
The search space is $W = Generations * |Population|$

4. Analysis

We conducted various experiments that would lead us to better understanding of the algorithm behaviour. Below we point how particular changes in the algorithm or in variables influences the final closeness of a result string to the pattern data.

We present data as following:

n	L	P_{cross}	P_{mut}	$\|Population\|$	Generations	No. of runs
number of elements in S	length of each string	crossover probability	mutation probability	size of each population	number of generations performed	Number of times the run was performed

The analysis are performed on the nucleotides alphabet.

4.1. Case 1: Applying schemata

n	L	P_{cross}	P_{mut}	$\|Population\|$	Generations	No. of runs
30	60	0.75	0.05	10	100	50

d()	without schema	with schema
28		1
29		9
30		26
31		14
...		
40	2	
41	6	
42	17	
43	18	
44	5	
45	2	

Conclusion
The profit from using scheme is significant. Counting schema is performed in $O((n + |Polulation|) * L)$ time, which does not extend the O() complexity of the whole algorithm.

4.2. Case 2: Various population sizes and generation numbers

number of elements in S: n = 30

n	L	P_{cross}	P_{mut}	$\|Population\|$	Generations	No. of runs
30	60	0.75	0.05	**P x .**	**. x G**	50

d()	10x10	10x100	100x10	100x100
27				2
28			1	1
29			9	13
30	6	2	26	29
31	23	12	14	5
32	33	13		
33	35	14		
34	2	6		
35	1	3		

number of elements in S: n = 10

n	L	P_{cross}	P_{mut}	$\|Population\|$	Generations	No. of runs
10	60	0.75	0.05	**P x .**	**. x G**	50

d	10x10	10x100	100x10	100x100
21				3
22	2		21	22
23	7	4	27	25
24	13	15	2	
25	19	23		
26	8	8		
27	1			

Conclusion:
As it could be expected, the run with the the population size 100 and number of generations 100 gives the best results.
Interestingly, the cases when the population is 10 and the number of generations is 100 gives worse results that the case where both values are 10. Therefore it's better to invest in the number of generations than in the number of population.

5. Conclusions

The experiments show, that with manipulating the variables in the Genetic Algorithm, we can perform different results in different time and space complexity. The most significant is the observation that it's better to invest in the number of generations than in the number of population.

Another observation leads to the conclusion, that in the field of CSP, using schema in generating an initial population of the Genetic Algorithm is recommended.

References

[1] Lusheng Wang Ming Li Bin Ma. "On The Closest String and Substring Problems". In: *Journal of the ACM (JACM)* 49 (2000), pp. 157–171.

[2] Faranak Bahredar et al. "A Meta Heuristic Solution for Closest String Problem Using Ant Colony System." In: *Advances in Intelligent and Soft Computing* 79.10 (2010), pp. 549–550. DOI: https://link.springer.com/chapter/10.1007/978-3-642-14883-5_70.

[3] Lushen Wang Zhi-Zhong Chena Bin Mab. "A three-string approach to the closest string problem". In: *Journal of Computer and System Sciences* 78 (2012), pp. 164–178.

[4] *biopp*. http://biopp.univ-montp2.fr/wiki/index.php/Main_Page. Accessed: 2017-09-15.

[5] *biopython*. http://biopython.org/. Accessed: 2017-09-15.

[6] *BLAST*. https://blast.ncbi.nlm.nih.gov/Blast.cgi. Accessed: 2017-09-15.

[7] *BLAT*. [http://genome.ucsc.edu/FAQ/FAQblat.html. Accessed: 2017-09-15.

[8] *EMBOSS*. http://emboss.sourceforge.net/. Accessed: 2017-09-15.

[9] *GenBank*. https://www.ncbi.nlm.nih.gov/genbank/. Accessed: 2017-09-15.

[10] Department of Computer Science San Jose State University Sami Khuri. *Genetic Algorithms. Lecture notes.* http://www.cs.sjsu.edu/~khuri/Aalto_2017/aalto_2017.pdf. Accessed: 2017-09-15.

[11] *Wikipedia. Hamming Distance*. https://en.wikipedia.org/wiki/Hamming_distance. Accessed: 2017-09-15.