Sampling Random Bioinformatics Puzzles using Adaptive Probability Distributions

Christian Theil Have Emil Vincent Appel
Jette Bork-Jensen Ole Torp Lassen

Novo Nordisk Foundation Center for Basic Metabolic Research, Section of Metabolic Genetics, University of Copenhagen, Denmark.

Roskilde University, Roskilde, Denmark

Probabilistic Logic Programming, 2016

Overview

This paper presents

- An application of Probabilistic Logic Programming (PRISM) to sample random bioinformatics puzzle games for educational purposes
- An approach we use deal with (avoid) failures during sampling.

Presentation outline

- A little background and motivation
- Just enough biology background to understand the concept of the game
- The game concept
- Sampling with constraints
- Sampling using adaptive probability distributions
- Discussion

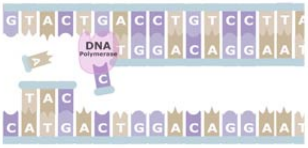
Motivation

We developed this game as part of workshop we need to explain to students from diverse backgrounds with bioinformatical understanding what Next Generation Sequencing is.

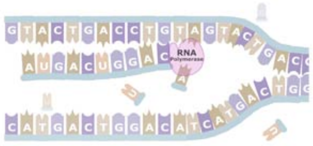
We wanted to make it fun and engaging and give students an impression of the algorithmic / bioinformatical challenges involved.

DNA, Proteins and the Central Dogma of biology

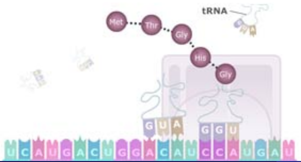
The Central Dogma of Biology



DNA Replication
After the separation of the two strands in the double helix, each strand is used as a template to make a new strand. All that's required are the proper nucleotides (A,T,C or G) in the proper pairing position.

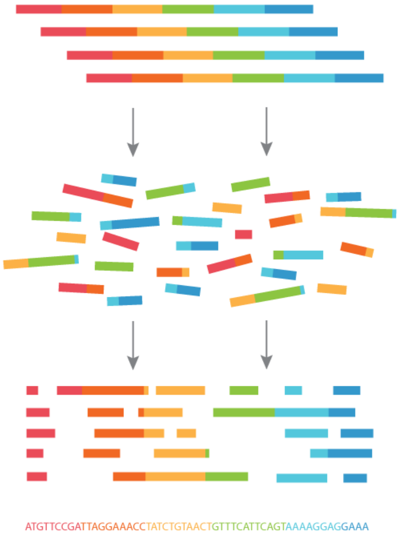


Transcription
To figure out how to build a protein, a copy of the genetic information is needed. We don't need the whole chromosome, just the gene of interest. So the cell makes a copy of the gene required and makes the new strand of genetic information out of RNA not DNA. Specifically, it makes an mRNA copy (messenger RNA) of the DNA sequence.



Translation Sequence
Ribosomes, messenger RNA (mRNA) and transfer RNA (tRNA) are involved in this multistep process. tRNA carries an amino acid to the ribosome, where the genetic code of the mRNA sequences the amino acids into polypeptides which are released into the cell and become

Next Generation Sequencing



Game background story

Recently an interesting protein with the amino acid sequence *ILP* was found in the bacteria *S. Equencia*. It is now to be determined if a homologue exists in the species *B. Ionformatica*.
To determine this a lab amplificied a relevant part of the DNA of *B. Ionformatica* using PCR primers flanking the gene in *S. Equencia* which are believed to be highly conserved also in *B. Ionformatica*, although the sequence of *B. Ionformatica* is currently not known. The amplified DNA was sequenced using Ullamini LoSeq next generation sequencing tech. The quality of the reads are not perfect – read errors resulting in random “mutations“ are expected in one out of twenty bases.

The challenge

As a bioinformatician you are given the task to find out if *B. Ionformatica* has a homologue of the protein *ILP* and determine how its amino acid sequence differs in *B. Ionformatica*. However, the high performance moon grid engine supercluster is currently down (as it sometimes is) and you have to do it all by hand. Fortunately, you have printed all the reads. You task is as follows:

1. Perform de-novo assembly of all the reads
2. Find open reading frames that may contain a gene
3. Find the amino acid sequence of any such gene to determine if it could be a homologue to *ILP*
4. Report your finding and claim eternal fame

The game board is empty to begin with

Amino acid sequence (forward strand)															
Nucleotide sequence (forward strand):															
A	**T**	**A**	**C**	**C**	**T**					**C**	**T**	**T**	**A**	**G**	**A**
Nucleotide sequence (reverse strand):															
T	**A**	**T**	**G**	**G**	**A**					**G**	**A**	**A**	**T**	**C**	**T**
Amino acid sequence (reverse strand)															

The reads are cut out

T	*A*	*T*	*G*	*G*	*A*	*A*	*A*	*T*	*G*
A	A	A	T	G	G	A	A	T	C
A	C	C	T	T	T	A	C	C	T
T	T	A	C	C	T	A	A	G	A
T	A	C	C	T	T	T	A	C	C
G	*G*	*A*	*A*	*A*	*T*	*G*	*G*	*A*	*A*
T	A	C	C	T	T	T	A	C	C
T	T	A	C	C	T	T	A	G	A
C	T	A	C	C	T	T	T	A	C

C	G	T	T	G	A	C	C	T	T

And placed on the board

Amino acid sequence (forward strand)															
Nucleotide sequence (forward strand):															
A	**T**	**A**	**C**	**C**	**T**					**C**	**T**	**T**	**A**	**G**	**A**
C	T	A	C	C	T	T	T	A	C						
T	*A*	*T*	*G*	*G*	*A*	*A*	*A*	*T*	*G*	C					
	T	A	C	C	T	T	T	A	C	C					
		A	C	C	T	T	T	A	C	C	T				
			C	G	T	T	G	A	C	C	T	T			
			G	*G*	*A*	*A*	*A*	*T*	*G*	*G*	*A*	*A*			
					T	T	T	A	C	C	T	T	A	G	
						T	T	A	C	C	T	T	A	G	A
						T	T	A	C	C	T	A	A	G	A
Nucleotide sequence (reverse strand):															
T	**A**	**T**	**G**	**G**	**A**					**G**	**A**	**A**	**T**	**C**	**T**
Amino acid sequence (reverse strand)															

Find consensus sequence

Amino acid sequence (forward strand)															
Nucleotide sequence (forward strand):															
A	**T**	**A**	**C**	**C**	**T**	T	T	A	C	**C**	**T**	**T**	**A**	**G**	**A**
C	T	A	C	C	T	T	T	A	C						
T	*A*	*T*	*G*	*G*	*A*	*A*	*A*	*T*	*G*	C					
	T	A	C	C	T	T	T	A	C	C					
		A	C	C	T	T	T	A	C	C	T				
			C	G	T	T	G	A	C	C	T	T			
			G	*G*	*A*	*A*	*A*	*T*	*G*	*G*	*A*	*A*			
					T	T	T	A	C	C	T	T	A	G	
						T	T	A	C	C	T	T	A	G	A
						T	T	A	C	C	T	A	A	G	A
Nucleotide sequence (reverse strand):															
T	**A**	**T**	**G**	**G**	**A**	A	A	T	G	**G**	**A**	**A**	**T**	**C**	**T**
Amino acid sequence (reverse strand)															

Use a codon table translate codons to amino acid

First base in codon	Second base in codon												Third base in codon
	T			C			A			G			
T	TTT	Phe	F	TCT	Ser	S	TAT	Tyr	Y	TGT	Cys	Y	T
	TTC	Phe	F	TCC	Ser	S	TAC	Tyr	Y	TGC	Cys	Y	C
	TTA	Leu	L	TCA	Ser	S	TAA	Stop	*	TGA	Stop	*	A
	TTG	Leu	L	TCG	Ser	S	TAG	Stop	*	TGG	Trp	W	G
C	CTT	Leu	L	CCT	Pro	P	CAT	His	H	CGT	Arg	R	T
	CTC	Leu	L	CCC	Pro	P	CAC	His	H	CGC	Arg	R	C
	CTA	Leu	L	CCA	Pro	P	CAA	Gln	Q	CGA	Arg	R	A
	CTG	Leu	L	CCG	Pro	P	CAG	Gln	Q	CGG	Arg	R	G
A	ATT	Ile	I	ACT	Thr	T	AAT	Asn	N	AGT	Ser	S	T
	ATC	Ile	I	ACC	Thr	T	AAC	Asn	N	AGC	Ser	S	C
	ATA	Ile	I	ACA	Thr	T	AAA	Lys	K	AGA	Arg	R	A
	ATG	Met	M	ACG	Thr	T	AAG	Lys	K	AGG	Arg	R	G
G	GTT	Val	V	GCT	Ala	A	GAT	Asp	D	GGT	Gly	G	T
	GTC	Val	V	GCC	Ala	A	GAC	Asp	D	GGC	Gly	G	C
	GTA	Val	V	GCA	Ala	A	GAA	Glu	E	GGA	Gly	G	A
	GTG	Val	V	GCG	Ala	A	GAG	Glu	E	GGG	Gly	G	G

Translating codons to amino acids

Amino acid sequence (forward strand)															
I/M			P			L			P			stop			
	Y			L			Y			L			R		
		T			F			T			L				
Nucleotide sequence (forward strand):															
A	**T**	**A**	**C**	**C**	**T**	T	T	A	C	**C**	**T**	**T**	**A**	**G**	**A**
C	T	A	C	C	T	T	T	A	C						
T	*A*	*T*	*G*	*G*	*A*	*A*	*A*	*T*	*G*	C					
	T	A	C	C	T	T	T	A	C	C					
		A	C	C	T	T	T	A	C	C	T				
			C	G	T	T	G	A	C	C	T	T			
			G	*G*	*A*	*A*	*A*	*T*	*G*	*G*	*A*	*A*			
					T	T	T	A	C	C	T	T	A	G	
						T	T	A	C	C	T	T	A	G	A
						T	T	A	C	C	T	A	A	G	A
Nucleotide sequence (reverse strand):															
T	**A**	**T**	**G**	**G**	**A**	A	A	T	G	**G**	**A**	**A**	**T**	**C**	**T**
Amino acid sequence (reverse strand)															
Y			R			N			G			I			
	M			E			M			E			S		
		W			K			W			N				

Identification of open reading frames

Amino acid sequence (forward strand)															
	I/M			P			L			P			stop		
		Y			L			Y			L			R	
			T			F			T			L			
Nucleotide sequence (forward strand):															
A	**T**	**A**	**C**	**C**	**T**	T	T	A	C	**C**	**T**	**T**	**A**	**G**	**A**
C	T	A	C	C	T	T	T	A	C						
T	*A*	*T*	*G*	*G*	*A*	*A*	*A*	*T*	*G*	C					
	T	A	C	C	T	T	T	A	C	C					
		A	C	C	T	T	T	A	C	C	T				
			C	G	T	T	G	A	C	C	T	T			
			G	*G*	*A*	*A*	*A*	*T*	*G*	*G*	*A*	*A*			
					T	T	T	A	C	C	T	T	A	G	
						T	T	A	C	C	T	T	A	G	A
						T	T	A	C	C	T	A	A	G	A
Nucleotide sequence (reverse strand):															
T	**A**	**T**	**G**	**G**	**A**	A	A	T	G	**G**	**A**	**A**	**T**	**C**	**T**
Amino acid sequence (reverse strand)															
	Y			R			N			G			I		
		M			E			M			E			S	
			W			K			W			N			

Final solution: Secret protein word identified

Amino acid sequence (forward strand)															
I/M			P			L			P			stop			
	Y			L			Y			L			R		
		T			F			T			L				
Nucleotide sequence (forward strand):															
A	**T**	**A**	**C**	**C**	**T**	T	T	A	C	**C**	**T**	**T**	**A**	**G**	**A**
C	T	A	C	C	T	T	T	A	C						
T	*A*	*T*	*G*	*G*	*A*	*A*	*A*	*T*	*G*	C					
	T	A	C	C	T	T	T	A	C	C					
		A	C	C	T	T	T	A	C	C	T				
			C	G	T	T	G	A	C	C	T	T			
			G	*G*	*A*	*A*	*A*	*T*	*G*	*G*	*A*	*A*			
					T	T	T	A	C	C	T	T	A	G	
						T	T	A	C	C	T	T	A	G	A
						T	T	A	C	C	T	A	A	G	A
Nucleotide sequence (reverse strand):															
T	**A**	**T**	**G**	**G**	**A**	A	A	T	G	**G**	**A**	**A**	**T**	**C**	**T**
Amino acid sequence (reverse strand)															
Y			R			N			G			I			
	M			E			M			E			S		
		W			K			W			N				

Implementation overview

- The program is implemented in the PRISM language
- The program executes in *sampling* mode to generate a *random* puzzle.
- The output of the program is a LaTeX document (deterministic)

Sampling and failures

In PRISM/PLP programs non-deterministic choices may be *random*, e.g.,

```
random_pair(X,Y) :-
    msw(dice,X),
    msw(dice,Y),
    X=Y.
```

This has procedural implications for the above (constrained) program if, e.g., X and Y differ (unification failure).

- In a Prolog program backtracking would in program order
- In PRISM sampling mode the we have committed random choices
- A unification failure (may) equal program failure

Constraints of the game

- The user can specify the secret protein word
- Protein word in background story must be a mutated version of the solution protein
- Number of mutations should be within a specified range
- Each position of the DNA sequence must be covered by a specified minimum number of reads (the depth)
- The maximal total number of reads is constrained (by board size)
- At any given position, the number mutations in reads covering the position should be less than or equal the number of non-mutated reads at that position

PRISM has an MSW construct that allows backtracing

```
random_pair(X,Y) :-
    soft_msw(dice,X),
    soft_msw(dice,Y),
    X=Y.
```

Problem solved?

PRISM soft msw implementation

```
soft_msw(Sw,Val) :-
    $pp_get_parameters(Sw,Values,Pbs),!,
    $pp_zip_vp(Values,Pbs,Candidates),
    $pp_soft_choose(Candidates,Val).

$pp_zip_vp([],[],[]).
$pp_zip_vp([Val|Vals],[Prob|Probs],[Val-Prob|Rest]) :- !,
    $pp_zip_vp(Vals,Probs,Rest).

$pp_soft_choose([],_V) :- !, fail.
$pp_soft_choose(Candidates,V) :-
    $pp_zip_vp(Vals,Probs,Candidates),
    sumlist(Probs,Sum),
    Sum > 0,
    random_uniform(Sum,R),
    $pp_choose(Probs,R,Vals,Val,Prob),
    delete(Candidates,Val-Prob,OtherOptions),
    (V=Val ; $pp_soft_choose(OtherOptions,V)).
```

Using soft_msw solves the problem of sampling with constraints:

```
random_pair(X,Y) :-
    soft_msw(dice,X),
    soft_msw(dice,Y),
    X=Y.
```

But.. Cronological backtracing incurs trashing: Repeated/redundant revisiting of derivation subtrees. This a problem when it is not the last choice point that leads to failure, e.g.,

```
yatzy(Score) :-
    soft_msw(dice,D1),
    soft_msw(dice,D2),
    soft_msw(dice,D3),
    soft_msw(dice,D4),
    soft_msw(dice,D5),
    soft_msw(dice,D6),
    sumlist([D1,D2,D3,D4,D5,D6],Score),
    0 is mod(Score,6).
```

Using adaptive probability distributions

- In our application, we experienced this problem when placing reads.
- The `soft_msw` approach didn't terminate in a timely fashing because of thrashing
- As an alternative/supplement to `soft_msw` we propose dynamicly adapting switch probabilities to avoid failures.
- In our application the adaptive probability distribution approach is much faster that using just `soft_msw`

Placement of reads

Reads are generated by a recursive predicate in which the termination case of the predicate specifies the condition that all positions must have the required minimum depth and the recursive case generates a random read, aligns it to the DNA sequence and updates a *depth vector*, $d_1 \dots d_n$, for each position $1 \dots n$ in the DNA sequence.

Placing reads - implementation in PRISM

```
placeread(Seq, Part1, Part2,ReadSize,Depths) :-
    length(Seq,L),
    LMax is L - ReadSize,
    findall(X,between(0,LMax,X),AllLen),
    findall(D2,(
        between(0,LMax,X),
        length(C1,X),
        length(C2,ReadSize),
        append(C1,C2,C3),
        append(C3,_,Depths),
        D2 is min(C2))),
    MinDepths),
    inverse_depths_probs(MinDepths,Probs),
    random_select(AllLen,Probs,L1),
    L #= L1+L2,
    length(Part2,L2),
    append(Part1,Part2,Seq).
```

Read placement probabilities

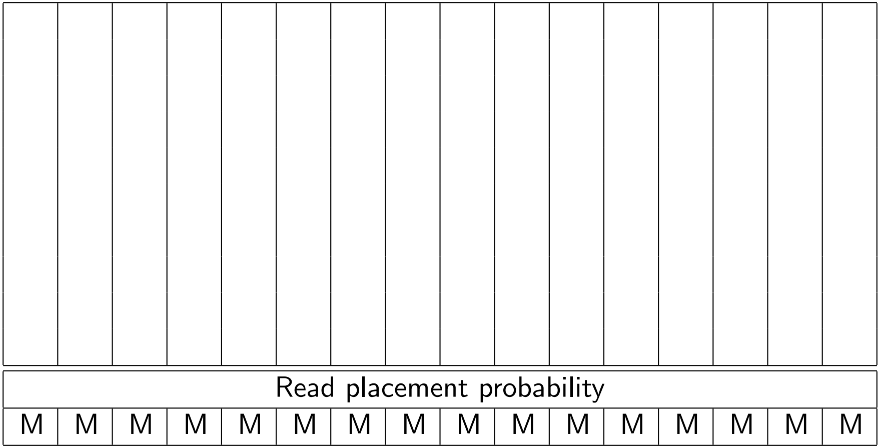
The probability of placing a new read of length r starting at position i, is given by,

$$P(pos = i) = \begin{cases} \frac{1}{n}, & \text{if} \sum_{i=1}^{n} d_i = 0. \\ \frac{w_i}{\sum_{h=1}^{n-r} w_h} & \text{otherwise.} \end{cases}, \text{ where } w_i = \frac{\sum_{j=1}^{n-r} \min d_j \dots d_{j+r}}{\min d_i \dots d_{i+r}}$$

Suppose for the sake of simplicity that we have only three possible probabilities, High (H), Medium (M) and Low (L).

Read placement probabilities 0

No reads placed yet → uniform probability



Read placement probabilities 1

First read placed

			C	G	T	T	G	A	C	C	T	T			

Read placement probability															
H	H	H	L	H	H	H	-	-	-	-	-	-	-	-	-

Read placement probabilities 2

Second read placed

			C	G	T	T	G	A	C	C	T	T			
		A	C	C	T	T	T	A	C	C	T				

Read placement probability															
H	H	L	L	H	H	H	-	-	-	-	-	-	-	-	-

Read placement probabilities 3

Third read placed

			C	G	T	T	G	A	C	C	T	T			
		A	C	C	T	T	T	A	C	C	T				
				G	T	T	T	A	C	C	T	T	A		

Read placement probability															
H	H	M	L	M	H	H	-	-	-	-	-	-	-	-	-

Conclusions

- We presented an puzzle game written using PLP
- We noted the thrashing problem when sampling from constraint PLP programs
- In the context of out game, we proposed using adaptive probability distributions to deal with this problem

Our approach has limitations:

- Specific to our application is not easily be generalized
- "Impure implementation" – other PRISM inferences are not possible with the program
- Distribution of depths/reads which appears uniformly random, but it is difficult to reason about properties of the distribution of successful derivations more generally

Points of future research

- Efficient sampling with constraints could be interesting for other applications, and if resulting probability distributions over possible derivations are accurate, sampling may be use a building blocks for more advanced inferences
- It would be useful to develop generic, but heuristically informed methods for PLP-based random sampling which are less prone to thrashing than the pure `soft_msw` backtracking approach. Perhaps inspiration can be drawn from the methods of constraint programming and intelligent backtracking