PLANT INSPIRED ALGORITHMS

I Plant Propagation Algorithm:

Inspiration for design of Optimization Algorithm.

1. Invasive Weed Optimization:

Algorithm has 3 components.

(i) Seeding:

Every plant produces a certain number of seeds. The number lies b/w N_{min}^{seeds} & N_{max}^{seeds}.

Each plant has a fitness value and this value ~~to~~ in comparison with the fitness values of all other plants in its surrounding gives the number of seeds it produces.

$$N_x^{seeds} = \frac{fitness_x - colfitness_{min}}{colfitness_{max} - colfitness_{min}} \left(N_{max}^{seeds} - N_{min}^{seeds}\right) + N_{min}^{seeds}$$

$fitness_x \rightarrow$ fitness of plant x

$colfitness_{min} \rightarrow$ minimum fitness of that plant colony

$colfitness_{max} \rightarrow$ maximum fitness of the plant colony.

(ii) Seed Dispersal:

Random displacement vector of dimension n

Find 'n' random values whose $\mu = 0$

$$\sigma_{iter} = \left(\frac{iter_{max} - iter_{curr}}{iter_{max}}\right)^n (\sigma_{max} - \sigma_{min}) + \sigma_{min}$$

σ decreases with each iteration, so does variance.

∴ Initial iteration → exploring the area

Later iteration → exploiting the same area.

(iii) Competition between plants:

At a time, in a given area only P_{max} number of plants can grow.

Wherever P_{max} value is reached, is the optimum solution.

2. Paddy Field Algorithm:

5 stages.

1. Sowing:

n number of plants

$x = (x_1 \; x_2 \ldots x_n)$

$y = f(x)$ is the fitness function.

2. Selection:

3. ~~Sam~~ Seeding:

$$S_i = f_{max}\left(\frac{y_i - y_t}{y_{max} - y_t}\right)$$

y_t = minimum fitness

y_{max} = maximum fitness.

4. Pollination:

$$U_i = e^{(V_i/V_{max})-1}.$$

$U_i \rightarrow$ pollination factor.

To measure neighbors:

a = radius.

number of plants in that radius are considered as neighbors.

$V_i \rightarrow$ neighbors of that plant

$V_{max} \rightarrow$ maximum number of neighbors (of any plant).

5. Dispersion:

$$S_i^{viable} = U_i S_i$$

DNA Computing Based Encryption & Decryption Algorithm

Requirements:

1. DNA Encoding of complete character set.
   - All the characters should have an encoding.
2. Dynamic encoding table generation.
   - Table for encoding & respective character should be different for each session
3. Unique sequence for encoding of every character.
   - All characters should have a unique encoding.
4. Robustness of Encoding.
   - Encryption should be random.
   - Difficult to decode
5. Biological Process Simulation
   - Every transformation should be a valid biological process
6. Dynamicity of encryption process.

Encoding:

Two encoding tables are generated using 2 intron sequences.

Encryption:

1. Plaintext converted into DNA sequence.
   - Divide plaintext into 2 halfs
   - Encode first half using encoding table 1 and second half using encoding table 2.
   - If the plaintext has odd number of characters, then add an extra character at the end of the string (as decided by the sender & receiver).

2. Multiple rounds of functions performed on the DNA sequence (minimum 10 rounds)
   a) DNA sequence XNOR with ~~DNA~~ introns.
   - First half with the first intron and 2nd half with the second intron.

Converting DNA to binary

A → 00

T → 01

C → 10

G → 11.

b) DNA converted to mRNA (transcription process)
T → U.
. Replace all Ts with Us.

c) mRNA to tRNA (translation process)
Complement
. Find the complement strand of given strand.

A → U
U → A
C → G
G → C.

} Complements.

d) tRNA to DNA (reverse transcription)
U → T
. Replace U with Ts.

e) Right Shift the sequences.

3. tRNA obtained is converted to amino acid

Q. Given the plain text BANK and the encoding for each letters as B → AAGG, A → ACAT, N → GCTT, K → GAGG. The intron 1 sequence is 01111000011111101 Intron sequence 2 → 10000111011111101 Encrypt the plain text, show only single round of multiple round functions.

BANK = AAGG ACAT GCTT GAGG

= 00001111 00100001 ; 11100101 11001111

01111000 01111101 ; 10000111 01111101

10001000 10100001 ; 10011101 01001101

CACA CCAG ; CTGT TAGT

Transcription:

CACA CCAG ; CUGU UAGU

Translation:

GUGU GGUC GACA AUCA

Reverse transcription:

GTGT GGTC GACA ATCA

Right shift:

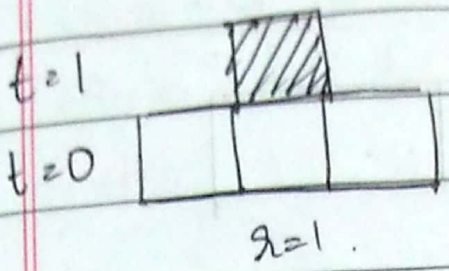
CGTG TGGT AGAC AATC

mRNA → CGUG UGGU ; AGAC AAUC

tRNA → GCAC ACCA ; UCUG UUAG

MB 13 ; WC W1

CELLULAR AUTOMATA

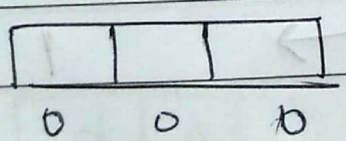


At t=0; If r = 1 (radius)
r cells on the left & r on the right
∴ neighborhood = 2r+1
= 2+1 = 3 cells

At t=1;
State change occurs.

Properties of cellular automata:
Homogeneity: Same set of rules
Parallelism: All cells = states updated simultaneously
Locality: Rules are local in nature

Q. Total number of ~~states~~ neighborhood states for r=1 & k=2.



k = state values
= 2 i.e 0 or 1.
r = radius ⇒ 3 cells in neighborhood

∴ Total number = 8.

$= 2^3$

$= k^{2r+1}$. (Generalized formula)

...	$C_{i-1}(t)$	$C_i(t)$	$C_{i+1}(t)$	...

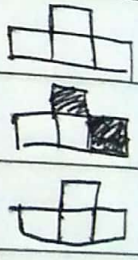
Transition function:

Example: $C_i(t+1) = [C_{i-1}(t) + C_i(t) + C_{i+1}(t)] \bmod 2$

For a 3 neighborhood (i.e 3 cells in the neighborhood)

The output can be 0 or 1 based on the values of $C_{i-1}(t)$, $C_i(t)$, $C_{i+1}(t)$.

$C_{i-1}(t)$	C_i	$C_{i+1}(t)$		$C_i(t+1)$
0	0	0	→	0
0	0	1	→	1
0	1	0	→	1
0	1	1	→	0
1	0	0	→	1
1	0	1	→	0
1	1	0	→	0
1	1	1	→	1



First order cellular automata:

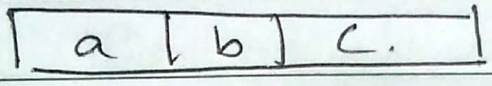
state at $t+1$ depends only on state at t.

Second order cellular automata

State at $t+1$ depends on state at t & $t-1$.

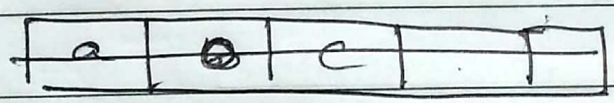
Boundary conditions:

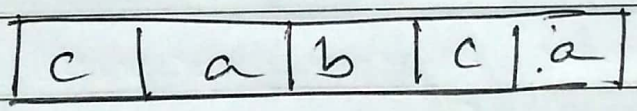
1. Periodic (wrap).



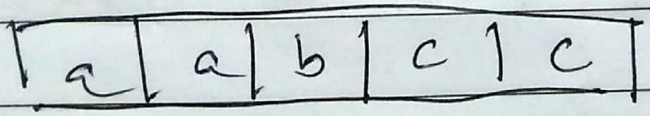
Boundary cells → cells without neighbors (a & c)

After wrapping.

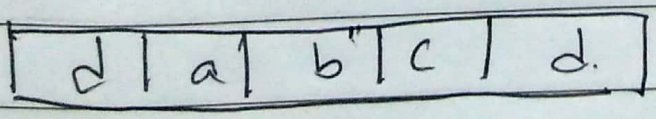




2. Reflective.



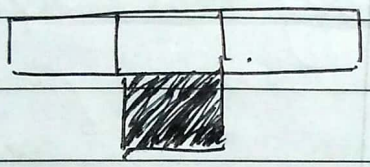
3. Fixed : Assume a fixed value for neighbor (say d)



Elementary Cellular Automata:

1. Rule 0 → produces 0.
2. Rule 1 → A neighbourhood of cells entirely in state 0 spawns live cell at the next time step.

t=0 t=1.



} The only condition to give a black / live cell.

All the other conditions give a white / dead cell.

3. Rule 30 → 011110

Rule 30 for Encryption System:

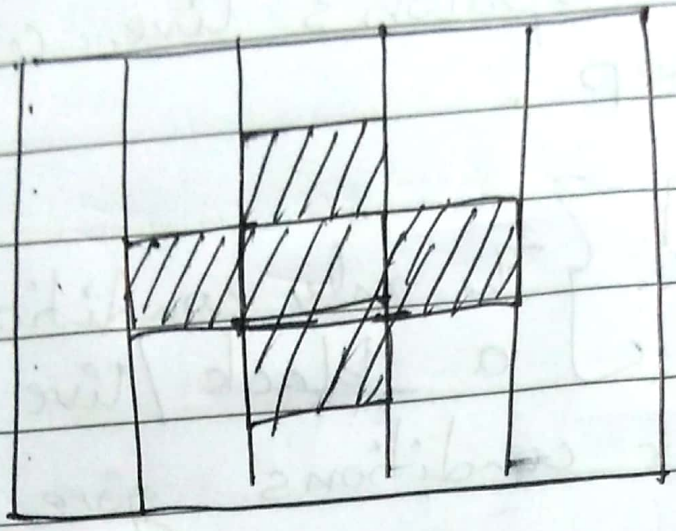
Steps for Encryption:

1. Plaintext to 8 bit ASCII (n x 8 array)

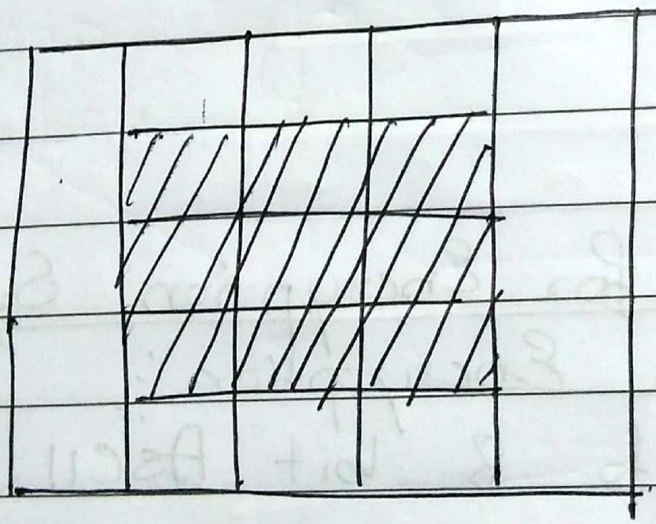
2-D Cellular Automata:

2 types of neighborhood.

1. Von - Neumann neighborhood: 4/5 cells
   $r = 1$.



2. Moore Neighborhood: 8/9 cells
   Diagonals also considered.



Extended Moore neighborhood $r > 1$.

Game of Life : Rules

A cell can die either of 'loneliness' (too less neighbors) or 'overpopulation' (too many neighbors).

- Dead cell with exactly 3 live neighbors → becomes a live cell.
- Live cell with 2 or 3 live neighbors → stays live.
- Live cell with $3 < n < 2$ live neighbors, dies.

Invariant forms : Configuration does not change with time.

Oscillators : Patterns that Oscillates.

DNA Based Archival Storage System

Bacteria is used since it's easy to manipulate it's & DNA

DNA Synthesis → Writing (Creating a DNA strand)

DNA Sequencing → Reading (Reading / finding nucleotide sequence).

Addressing.

Each DNA strand has a forward & a backward primer.

Compute the key using these primers. Then map it to a DNA pool according to the key.

From Huffman code (Base 3) to DNA nucleotides.

Rotating encoding.

	A	C	G	T
0	C	G	T	A
1	G	T	A	C
2	T	A	C	G

1 2 0 1 1. → G C G A C G.

↓

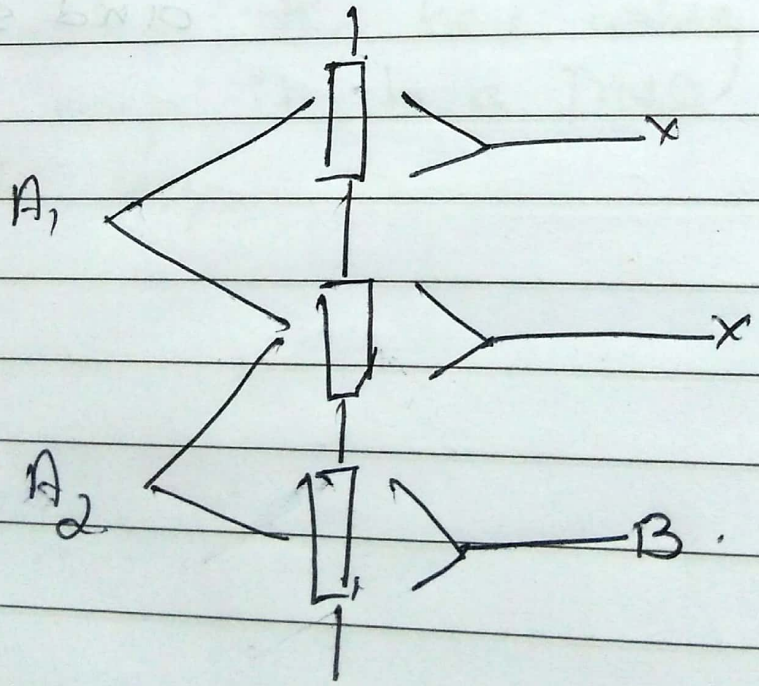
G (1st column)

Since previous is G, 2nd row G[th] column ⇒ C.

and so on...

Peptide Computing:

Peptides have epitopes (recognition sites).
Antibodies can attach to epitopes.
Multiple antibodies can ~~attach~~ try to attach to the same epitope.
But each antibody has an affinity to the epitope. Whichever has the higher affinity has higher priority to attachment. (It can even preempt the lower affinity antibody that is already joined at the epitope).



Affinity: $A_1 < x < A_2 < B$.

- Use antibody A_2 and Set G is added.
  ∴ All X's from G get appended at the first position.
- Add antibody B, it removes A_2.
  Add Set H.
  ∴ All X's from H get appended at 2nd position.
- A1 is added to set all elements.
- Add labeled element. Gets attached at either 1st or 2nd position, which ever is free.

The labelled element emits ~~light~~ color.

∴ If 2nd position emits color, then the number of X's in G is higher.

~~If 1st position emits color, then the number of X's in H is higher.~~

If No color is detected then, H has at least as much as G & more number of X's.

Emit color ⇒ G > H.

No color ⇒ H ≥ G.