Turing Machines and some applications

Diogo Pratas

IEETA/LASI/DETI,
University of Aveiro, Portugal
pratas@ua.pt

Contents

Contents

What is a Turing Machine?

A **Turing Machine** is a theoretical computing model that consists of:

- An **unlimited tape** that acts as memory.
- A **tape head that moves** left or right across the tape.
- A **set of states** that control the machine's behavior.
- A **set of rules** (transition functions) that dictate how the machine moves and alters the tape based on the current state and symbol under the tape head.

It is used to **model any computation that can be performed by a computer**, providing a formal definition of algorithmic processes.

Purpose and Significance of Turing Machines

- **Computational Theory:** Turing Machines help define what can be computed in principle and lay the foundation for the study of computation.
- **Church-Turing Thesis:** They are equivalent in computational power to other models of computation, such as Lambda Calculus or modern digital computers.
- **Decidability and Computability:** Turing Machines are central to understanding which problems are solvable (decidable) and which are not (undecidable).
- **Universal Turing Machine:** It can simulate any other Turing Machine, demonstrating the concept of universal computation.

History of Turing Machines

The concept of the Turing Machine was introduced by **Alan Turing** in 1936 in his groundbreaking paper, “On Computable Numbers, with an Application to the Entscheidungsproblem.”

- **1936:** Turing’s original work laid the foundation for modern computer science.
- **1936-1937:** Turing introduced the notion of the Universal Turing Machine, a machine capable of simulating any other Turing Machine.
- **1940s:** Turing’s ideas contributed to the development of early digital computers.
- **Post-WWII:** Turing’s concepts played a critical role in the theoretical and practical development of computing.

Turing’s work became one of the most important contributions to the field of theoretical computer science.

Turing Machine Transition Table – Two States, Two Symbols

Read	State A			State B		
Symbol	Write	Move	Next	Write	Move	Next
0	1	$>$	B	1	$<$	A
1	0	$<$	B	0	$>$	.

* "$>$" is move right, "$<$" is move left, and "." indicates machine stops.

Turing Machine Transition Table – Busy Beaver (3 states, 2 symbols)

Read	State A			State B			State C		
Symbol	Write	Move	Next	Write	Move	Next	Write	Move	Next
0	1	$>$	B	1	$<$	A	1	$<$	B
1	1	$<$	C	1	$>$	B	1	$>$	.

* "$>$" is move to right, "$<$" move to left, "$=$" do not move, and "." halt.

A **Busy Beaver** is a theoretical concept in computer science and computability theory that refers to a Turing Machine designed to do "as much as possible" before halting (Tibor Radó, 1962).

Contents

Contents

https://github.com/cobilab/spark

Simulation and search for exact or approximate Turing Machines

This program schools, simulates, and searches for exact or approximate Turing machines (TMs) with specific characteristics. It uses alignment-free approaches for searching the tapes and ascii color for beter understanding. Time (sleep) changes are flexible and well as modes.

Install SPARK:

```
git clone https://github.com/cobilab/spark
cd spark/src/
cmake .
make
./SPARK -h
```

```
(base) x@arvis:~/git/spark/src$ ./SPARK -1
Mode             :  1
Using seed       :  1741694894
Init state       :  3
Max time         :  10000
Max amplitude    :  20000
Min amplitude    :  50
Visual delay     :  50000
Alphabet         :  A, B, C, D
States           :  0, 1, 2, 3, 4, 5, 6
Actions          :  <, >, =

             0        1        2        3        4        5        6
 A       A > 6    C = 5    A > 3    B > 1    B < 0    D = 4    D = 1
 B       B = 5    D < 4    B > 3    B = 1    A < 1    B < 5    A > 1
 C       D = 5    D > 3    A < 6    D > 4    C = 4    B < 3    B = 2
 D       C < 2    C < 1    A = 4    A < 0    D > 0    B = 1    A < 5

Time [108] | Tape: BDADADADADADADADADADADADADADADADADADADADCBAAAAAAAA
```

```
(base) x@arvis:~/spark/src$ ./SPARK --mode 1 --alphabet ABCD --alphabet-size 3 --states-number 10 -
-seed 7 --rand-type 1 --delay 100000 --halt --max-time 100
Mode            :  1
Using seed      :  7
Init state      :  7
Max time        :  100
Max amplitude   :  20000
Min amplitude   :  50
Visual delay    :  100000
Alphabet        :  A, B, C
States          :  0, 1, 2, 3, 4, 5, 6, 7, 8, 9
Actions         :  <, >, =, .

          0       1       2       3       4       5       6       7       8       9
 A      A = 1   C < 6   B < 3   C > 1   A > 0   B = 0   A = 3   A . 9   B > 2   B = 3
 B      B < 2   A = 7   A > 8   B > 9   A = 7   A > 5   A > 1   A > 5   B < 6   A = 8
 C      A < 8   B > 6   B = 1   A = 3   B = 1   B > 0   A > 3   A = 6   B > 6   B = 1


This machine has halted (.)
```

- `--states-number` 10 and `--alphabet-size` 4
- `--halt` vs `--max-time` 100
- `--seed` 7, `--rand-type` 1, `--delay` 100000

Contents

Contents

Complex Behaviour Tapes:

./SPARK `--states-number` 16 `--output-top` top.txt `--mode` 3 `--alphabet` ACGT `--alphabet-size4` `--seed` 7 `--rand-type` 1 `--max-time` 5000 `--machines` 1000000 `--threads` 4 `--verbose` `--threshold` 0.90 `--min-amplitude` 100 `--max-amplitude` 2000

Look for:

more top.txt-1.inf
more top.txt-2.inf
more top.txt-3.inf
more top.txt-4.inf

Contents

Contents

Can we compress this sequence?

```
CCGATTATTTAGGCTATTAGATTATTAAAAGGTTAGGCTGCTAGGATGGAAGATGGTGCTATCTATTGTT
AGGCTATTAAAGAAGAAAGAAAAAAAAGTCTAGCCACTGCCAGTTCCTGTTCTGCACGAGTGCACGGGCTC
ACACTGCTACGCCGAGCCGGCTGCTTACTGCGCACCTCCTGGTACTGCCGACTGTAGCCGGCTGCGGCGG
TACTGCTACCCACTGTCGCGCACGGCGTACGGTCGCACGAGTTCCTCACTTACTGCGACGCCGGCCACAC
TGCCGCACGTGCACAGCCGGCGGTCTCACTTTCCACTGTACGGTCTCACTTTCCTGTACACGGTCGG
```

Data Compression of Biological Sequences

- **GeCo Series:** Advanced genomic compression tools such as GeCo3, integrating neural networks for enhanced efficiency in reference-free and reference-based DNA compression.
  `https://github.com/cobilab/geco3`

- **JARVIS Series:** Reference-free genomic compression tools excelling in handling complex structures like inverted repeats. JARVIS3 optimizes memory usage, speed, and supports FASTA/FASTQ formats.
  `https://github.com/cobilab/jarvis3`

- **AC Series:** Proteome compression tools introducing neural network enhancements (AC, AC2), achieving significant compression ratios for large-scale protein datasets.
  `https://github.com/cobilab/AC2`.

```
CCGATTATTTAGGCTATTAGATTATTAAAAGGTTAGGCTGCTAGGATGGAAGATGGTGCTATCTATTGTT
AGGCTATTAAAGAAGAAAGAAAAAAAAGTCTAGCCACTGCCAGTTCCTGTTCTGCACGAGTGCACGGGCTC
ACACTGCTACGCCGAGCCGGCTGCTTACTGCGCACCTCCTGGTACTGCCGACTGTAGCCGGCTGCGGCGG
TACTGCTACCCACTGTCGCGCACGGCGTACGGTCGCACGAGTTCCTCACTTACTGCGACGCCGGCCACAC
TGCCGCACGTGCACAGCCGGCGGTCTCACTTTCCACTGTACGGTCTCACTTTCCTGTACACGGTCGG
```

- Assuming independence: $347 \log_2(4) = 694$ bits
- GeCo3 (with optimization) achieved 683 bits (excluding header)
- GeCo3 was only able to compress 11 bits (from 694)
- The data seems to be close to random (high-complexity sequence)
- Hairpin pattern

Symbols \ States	0	1	2	3	4	5
A	A > 4	T < 4	T > 4	A < 1	C < 2	A > 2
C	T > 3	T > 2	G < 3	G < 1	G > 4	C < 4
G	C < 4	C < 0	T < 1	A > 1	G < 3	A = 2
T	C > 0	C < 3	G > 3	C < 3	C > 2	A > 5

Actions:
- \> move right
- = no move
- < move left

From a “blank” tape (all to A)

Inverse Challenge

- **Turing Machines**
- **Alignment-free**
- **Tolerant Model + side information**
- **Multi-threading**
- **Complex string inputs**
- **Optimization**

Computation: −−mode 4

Questions

- Given a complex (algorithmically random) string, is it possible to construct a Turing Machine that generates this string on its tape and eventually halts?
- In what ways can the initial configuration of the input tape affect the process of discovering a complex string in a Turing Machine computation?
- For every complex (algorithmically random) string, is there a minimal Turing Machine that can generate the string and halt?
- How can the SPARK framework be used to approximate the logical depth of a given computation?
- How can the SPARK framework be optimized in this search?

Q&A

pratas@ua.pt

https://pratas.github.io

https://github.com/cobilab

https://github.com/viromelab