n-Skip Turing Machines

Wiktor K. Macura[*]
Department of Mathematics and Statistics,
University of Maryland, Baltimore County,
Baltimore, Maryland, 21228

A Turing Machine's head is limited to moving one cell in either direction on the tape for a given iteration. We investigate a form of Turing Machine where the head is allowed to move n cells in either direction. We find that such Turing Machines, named n-Skip Turing Machines, are capable of exhibiting complex behavior for simple initial conditions with two states and two colors.

1. Introduction

The Turing Machine was designed by Alan Turing to serve as a general computational model. The machine has a tape to which data can be written by a head. A tape consists of cells where each cell has a value (similar to a cell on a hard-drive, albeit the cells of a Turing Machine's tape can have more than two states) and that value can be changed by the head. The head can also be in multiple states and it may move. What the head writes to the tape for a given iteration is determined by the value (color) of the cell and the state of the head. For a given case, the cell's color, the head's state, as well as the head's position may change. Of particular note, however, is that the head may only move by one cell in either direction [1].

A Turing Machine can be made to exhibit complex behavior for simple initial conditions by adding states, colors, or both. In [2] Wolfram shows that the simplest Turing Machine with complex behavior with an empty tape as input has four states and two colors. What we consider is whether one can create a Turing Machine which exhibits complex behavior by letting the head move by more than one cell in either direction (we assume a one-dimensional tape).

We define such a machine to be a n-Skip Turing Machine, where the n refers to the maximum number of cells a head can skip for a given iteration. Notice that we do not impose the limitation that the head must move n cells.

We define the rule of a n-Skip Turing Machine as follows (with two states and two colors):

[*]Electronic mail address: wmacur1@umbc.edu.

$$\{T_1, X_1\} \to \{\tau_1, \chi_1, \kappa_1\} \quad \{T_1, X_2\} \to \{\tau_2, \chi_2, \kappa_2\}$$
$$\{T_2, X_1\} \to \{\tau_3, \chi_3, \kappa_3\} \quad \{T_2, X_2\} \to \{\tau_4, \chi_4, \kappa_4\}.$$

Where T is the state and X the color of the cell at which the head is positioned. Similarly, τ is the state and χ the color of the new cell, and κ is the new position of the head. Note that the number of cases is independent of the size of the skip, in particular, there are $t \times k$ cases with t states and k colors.

Obviously, one cannot have a n-Skip Turing Machine with a negative n.

Generalizations

One can write a rule for a particular n-Skip Turing Machine (with t states and k colors) as:

$$\begin{array}{ccc} \{T_1, X_1\} \to \{\tau, \chi, \kappa\} & \ldots & \{T_1, X_k\} \to \{\tau, \chi, \kappa\} \\ \vdots & \ddots & \vdots \\ \{T_t, X_1\} \to \{\tau, \chi, \kappa\} & \ldots & \{T_t, X_k\} \to \{\tau, \chi, \kappa\}. \end{array}$$

Notation

We use notation of the form `{36023551, 12}` to mean rule number 36023551 with a skip size of 12 (we only use two states and two colors, throughout). See section 5 for the function `nRule` which creates the rule from the number and skip size.

2. Rule count per skip size

There exists $(4 \times (2n + 1))^4$ rules for a skip size of n (note that we count cases which result in a skip size of 0). More generally, for t states and k colors we have ω rules where

$$\omega = (tk(2n + 1))^{tk}.$$

If we do not include cases with a skip size of 0 we have ω_2 rules where

$$\omega_2 = (2tkn)^{tk}.$$

This gives the following small table of rule counts for skip sizes ranging from 0 to 50 (note that $t = k = 2$ in the table).

n	ω	ω_2
0	256	0
1	20,736	4096
2	160,000	65,536
3	614,656	331,776
4	1,679,616	1,048,576
5	3,748,096	2,560,000
10	49,787,136	40,960,000
50	26,639,462,656	25,600,000,000

3. Typical behavior

We are interested in whether adding a "skip" to the head of a Turing Machine can create complex behavior in itself. As such, we only look at *n*-Skip Turing Machines where there are two colors and two states, the simplest example of a Turing Machine that can generate complex behavior (albeit only with complex input). Similarly, our initial condition is very simple: an empty tape.

A sample output from a 5-Skip Turing Machine is provided in Figure 1.

One generally finds four types of output when increasing the skip size.

- Output that can be created with a machine of lesser skip size. This results from the rules of a *n*-Skip machine being a subset of the rules of a $(n+1)$-Skip machine.
- In a variation of the above, one can find rules that "stretch" the output.
- One can find output that is slightly different, but has the same general look.
- Finally, one can find totally new output.

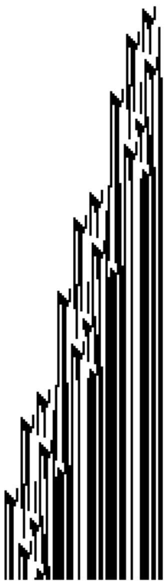

Figure 1. An example of a *n*-Skip Turing Machine. This one does not show complex behavior, but notice that it is showing behavior that is somewhat complex but then repeating it, a trait that is very common in *n*-Skip Turing Machines `{161073, 5}`.

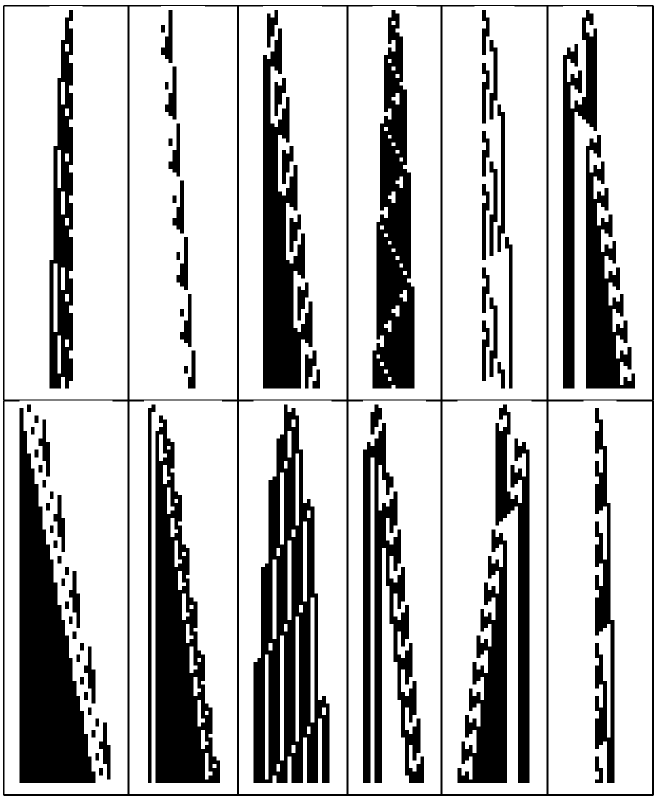

Figure 2. A gallery of various n-Skip Turing Machines. All output is shown after 100 iterations.

The machines' outputs were evaluated by finding their evolution through 50 iterations, and then running a `Union` on the output to remove machines with the same output. Finally, the output was checked for complexity manually.

0-Skip Turing Machines

We add these only for completeness. There are 256 of these machines, 16 generate unique output.

1-Skip to 8-Skip Turing Machines

Skip sizes less than nine do not result in complex output. Although one finds it gets increasingly complex before becoming repetitive or nested. Figure 2 provides a gallery of various n-Skip Machines with $2 \leq n \leq 8$.

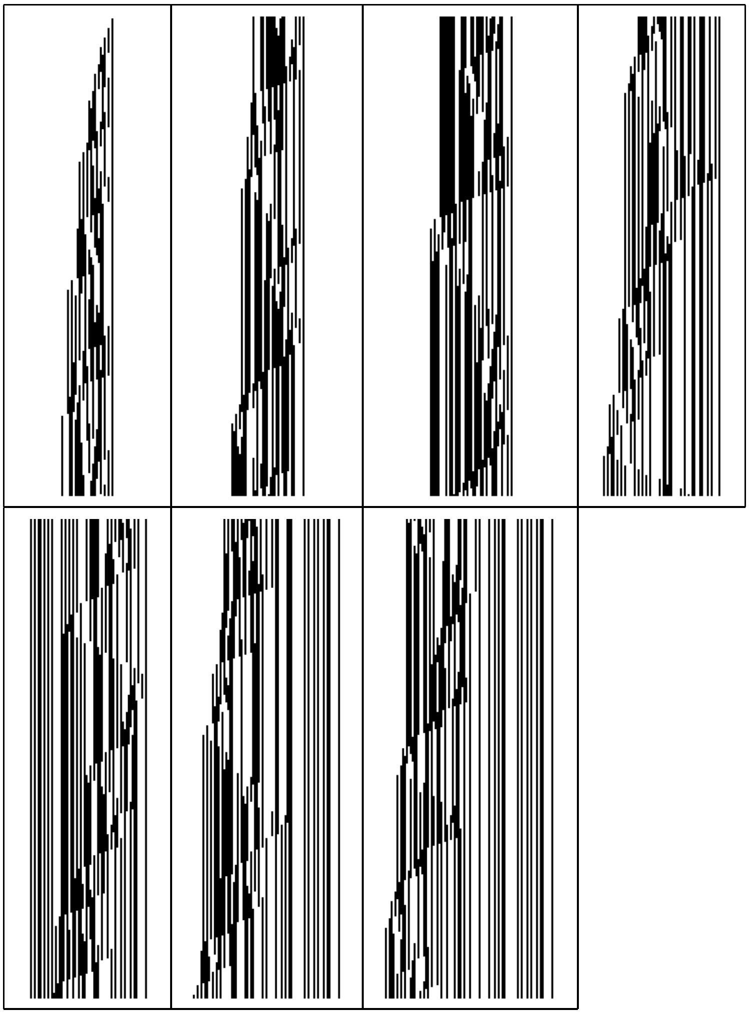

Figure 3. A nice example of a 9-Skip Turing Machine which exhibits complex behavior. Each of the seven boxes shows 250 steps in the evolution of the rule $\{\{2,1,9\},\{2,0,-3\},\{1,0,-3\},\{1,1,-2\}\}$.

9-Skip Turing Machines

9-Skip Turing Machines are the first machines which exhibit complex behavior. See Figure 3 for an example.

Why 9-Skip Turing Machines create complex output would require more research. However, one can surmise that interference has a large effect. In the first dozen iterations the machine is well-behaved. Yet then the first case is triggered, which causes the head to jump to the right by nine cells. At that point, the head starts processing the data that was written beforehand, resulting in increasingly chaotic behavior.

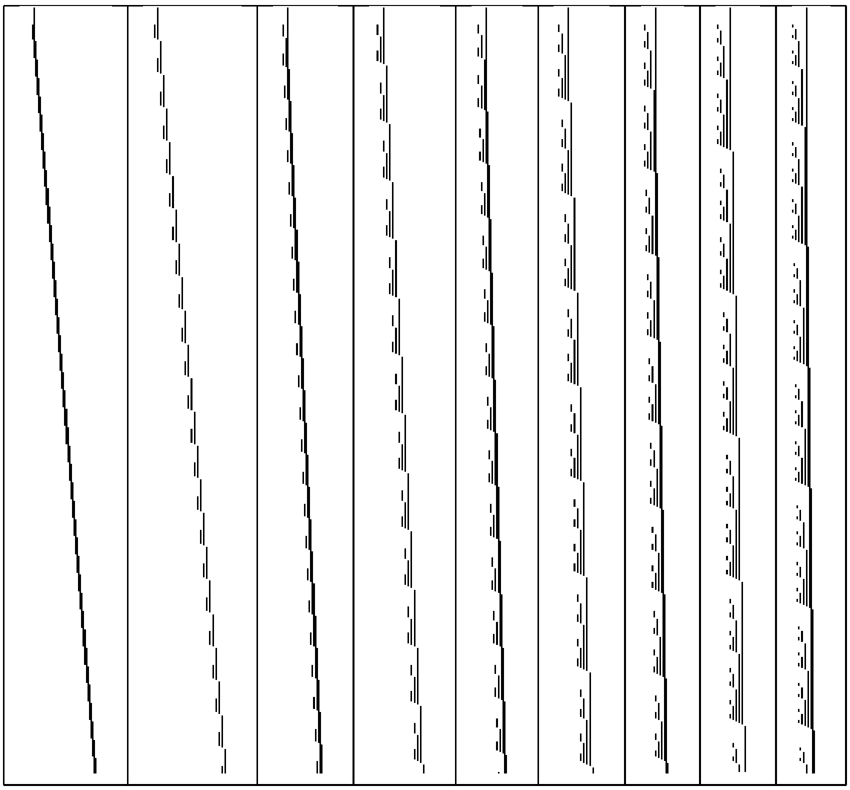

Figure 4. The effect of changing the skip size of one case in a rule from −1 through to −9.

4. Effects of changing the skip size

Not surprisingly, changing the size of a particular skip within a rule generally stretches the rule in the respective direction (see Figure 4).

It is worth noting that complex behavior usually occurs when a single case results in a relatively large skip size and the other cases have relatively small skip sizes. The reason for this appears to be that complex behavior occurs when there is large interference with the head, in other words, when the head keeps going over nonempty cells. The need for a large skip size is, however, puzzling.

5. Source code

The following *Mathematica* [3] code generates the two-dimensional representation of the tape.

```
step[rl_, {s_, t_, p_}] :=
  With[{u = {s, t[[p]]} /. rl}, {u[[1]],
    ReplacePart[t, u[[2]], p], p + u[[3]]}]

stepres[{s_, t_, p_}] :=
  If[p < 1, {s, Join[Array[0 &, 1 - p], t], 1},
    If[p > Length[t], {s, Join[t, Array[0 &, p - Length[t]]], p},
{s, t, p}]]

evo[rl_, k_, n_] := NestList[stepres[step[rl, #]] &,
  {1, Array[0 &, 2n + 1], n + 1}, k]

adjres[rl_, {s_, t_, p_}] :=
  With[{u = {s, t[[p]]} /. rl}, {Max[0, 1 - (p + u[[3]])],
  Max[0, p + u[[3]] - Length[t]]}]

adjevo[rl_, hist_] :=
  MapThread[Join[Array[0 &, #1], #2, Array[0 &, #2]] &,
  {#[[-1, 1]] - First /@ #, #[[2]] & /@ hist, #[[-1, 2]]
  - Last /@ #}, 1] &[Most[FoldList[Plus, {0, 0},
  adjres[rl, #]& /@ hist]]]

evolution[rl_, k_, n_] := adjevo[rl, evo[rl, k, n]]
```

The proper usage is to call `evolution` with a rule (formatted as in section 1), `rl`, the number of iterations `k`, and the largest skip size in the rule `n`. Note that one can additionally use the following code to avoid specifying the largest skip size.[1]

```
nskip[rl_, k_] :=
  evolution[rl, k, Max[Abs[#2[[3]]] & @@@ rl]]
```

`nRule` takes a number and a skip size (and, optionally, the number of colors and states) and generates the rules to be used with `nskip` or `evolution`.

```
nRule[ruleNum_, skip_:9, colors_:2, states_:2] := (
 rule = ruleNum*states;
 a = Table[{d, c} -> Reverse[{
   (Mod[(rule = Floor[rule/states]), (2 skip+1)]-skip),
   (Mod[(rule = Floor[rule/(2 skip + 1)]), colors]),
   (Mod[(rule = Floor[rule/colors]), states] + 1)
         }]
  , {d, states}, {c, 0, colors - 1}];
 Flatten[a]
)
```

6. Future work

It would be interesting to find out why nine is the "magic" number for getting complex *n*-Skip Turing Machines as well as the possibilities of *n*-Skip Turing Machines when given complex input. It would also be nice to research why the pattern mentioned results in complex behavior, specifically, why such a large skip is needed.

[1]The largest skip size is necessary for the `evolution` code as it is used to create the initial tape which must be long enough to not have to worry about overflows.

7. Acknowledgments

I would like to thank Ed Pegg and Todd Rowland for providing invaluable assistance during the course of this project at the New Kind of Science Summer School and also afterwards.

I would like to thank Stephen Wolfram for suggesting the idea of research *n*-Skip Turing Machines.

In addition, I thank all of the students and staff at the New Kind of Science Summer School for providing camaraderie and a productive working environment.

References

[1] Eric W. Weisstein. “Turing Machine.” From *MathWorld*–A Wolfram Web Resource. http://mathworld.wolfram.com/TuringMachine.html

[2] S. Wolfram, *A New Kind of Science* (Wolfram Media, Inc., Champaign, IL, 2004)

[3] Wolfram Research, Inc., *Mathematica*, Version 5.1, Champaign, IL (2004).