Machine Learning Project Proposal
Evolutionnary Computing on Turing machines

Jean-Florent Raymond
`Jean-Florent.Raymond.8795@student.uu.se`

April 20, 2012

1 The Busy Beaver

Introduction Given an number of states and a number of symbols, a *busy beaver* is a deterministic Turing machine that, when started on a blank tape, performs "the most operations" before halting among all Turing machines with the same numbers of states and symbols. Performing "the most operations" can be definded in many ways, for instance it can mean

- doing the most steps ;
- stopping with the most non-blank symbols written on the tape.

All the Turing machines we will consider here are deterministic. Let $\mathcal{T}_{m,n}$ be the set of deterministic Turing machines with an alphabet of m symbols who uses n states and $\mathcal{T} = \cup_{n,m \in \mathbb{N}} \mathcal{T}_{n,m}$.

Busy Beaver and Max-step Functions Let $\sigma : \mathcal{T} \to \mathbb{N} \cup \{-\infty\}$ be the function who, given a Turing machine M, returns

- either the number of non-blank symbols that M, started on a blank tape, leaves on the tape after execution if M halts ;
- or minus infinity else.

Let the *busy beaver function* Σ be defined as follows

$$\Sigma : \left| \begin{array}{l} \mathbb{N} \times \mathbb{N} \to \mathbb{N} \\ (m, n) \mapsto \max_{M \in \mathcal{T}_{m,n}} \sigma(M) \end{array} \right.$$

Similarly, let $s : \mathcal{T} \to \mathbb{N} \cup \{-\infty\}$ be the function who, given a Turing machine M, returns

- either the number of steps that M performs when started on a blank tape, if M halts ;
- of infinity else.

And let *max-step function* S be defined by

$$S : \left| \begin{array}{l} \mathbb{N} \times \mathbb{N} \to \mathbb{N} \\ (m,n) \mapsto \max_{M \in \mathcal{T}_{m,n}} s(M) \end{array} \right.$$

Note that these four function are not computable because the opposite case would means that we can decide the halting problem [?].

2 Finding Machines That Reaches Bounds

The goal of this project is to find, at least for small values of m and n, Turing machines that reaches $\Sigma(m,n)$ or $S(m,n)$, *i.e.* finding a machine $M \in \mathcal{T}_{m,n}$ such that $\sigma(M) = \Sigma(m,n)$ or $s(M) = S(m,n)$.

Such machines have already been found [?] but only for small values of m and n because S and Σ functions grows very fast [?], what makes tests difficult.

How ? This will be done using evolutionary computing techniques on a population of Turing machines comming from the same class $\mathcal{T}_{m,n}$, for fixed m and n. Operations of crossover and mutations are made from the transition tables (the alphabet and the number of states will remain unchanged).

Fitness Function The main problem is to define a good fitness function. The value to maximize here is the number of steps before halting (for the max-step case), in order to reach the bound.

But we cannot says that between a machine M which halts after p steps and a machine M' which continues to run after p steps, M' is better w.r.t. max-step because we don't know if M' will stop and we have no general way to know it (since the halting problem is undecidable [?]).

However, for fixed $p \in \mathbb{N}$, we are able to say if a given machine will stop in less than p steps. To know it, we only run the machine during p steps and watch if it is on a halting state at this time.

The approach we want to try here is depends on what bound we want to reach :

- for small values of $n, m \in \mathbb{N}$, values of Σ and S are known [?] so a suitable fitness function is a function who, given a Turing machine M that we already run on p steps without leading it in the halting state, returns $-\infty$ if $p > S(m,n)$ and $\frac{1}{S(m,n)-p}$ else. With this function, Turing machines which overtake the bound are highly penalized (because we know they will never halt) and a machine who run longer (but less than $S(m,n)$) without halting than an other one get a better grade.

- for values of $n, m \in \mathbb{N}$ for which Σ and S are not known, we can, for growing values of $p \in \mathbb{N}$, find machines who halts after p steps with a fitness function of the same kind as before. Then the more p will be high, the more we will approximate the bound if we still find machines for this p.

3 Work Packages

The project will be splitted in several work packages :

1. Virtual Machine
   - defining a convenient way to represent Turing machines ;
   - writing a basic virtual machine to run Turing machines ;
   - adding the ability to load and save machine states in a file and to log activity ;
   - adding abilities of step counting and of running until the machine halts (if it halts).
2. Evolutionary Computing Tool
   - defining operations on Turing machines (crossover, mutations) ;
   - defining a fitness function ;
   - writing a program who generates a random initial population and who follows it until it leads to something interesting.
3. Testing
   - testing all the parts of the program we write the ensure that we will not do hours of useless computation during the experiments ;
   - launching the experiments.
4. Analyse
   - analysing the results : Do we found what we were looking for ? Do we discover other things ?
   - analysing the approach : Was evolutionary computing a good idea for this problem, and why ?
5. Writing documentation.

The schedule for this project can be found on figure 1, page 4.

4 Related Work

The work of Shen Lin and Tibor Rado on busy bever was based on reducing the space search with normalization techniques and then on examining patterns on the tape [**?**].

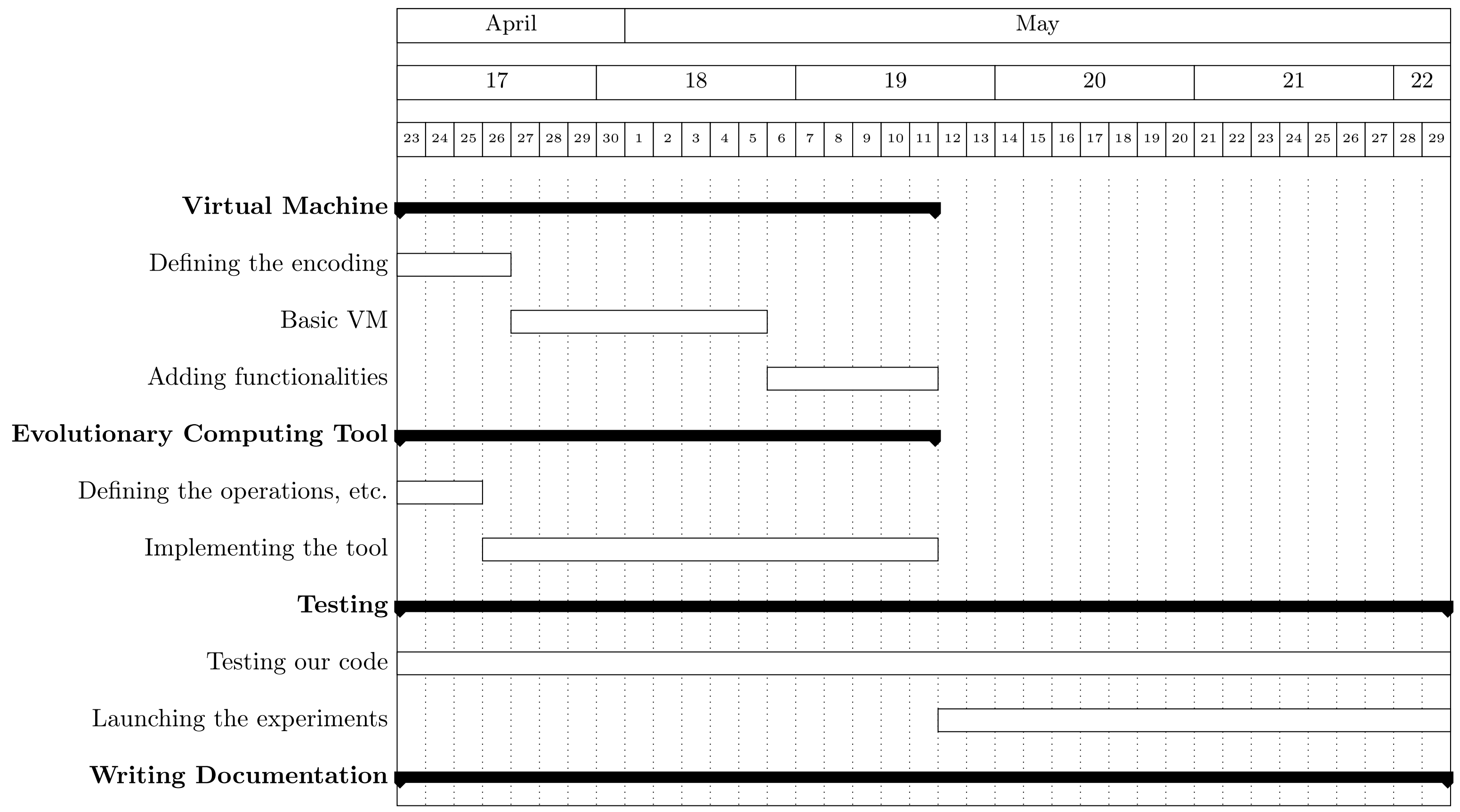

Figure 1: Schedule