Bioinformatics Final Project, Fall 2000

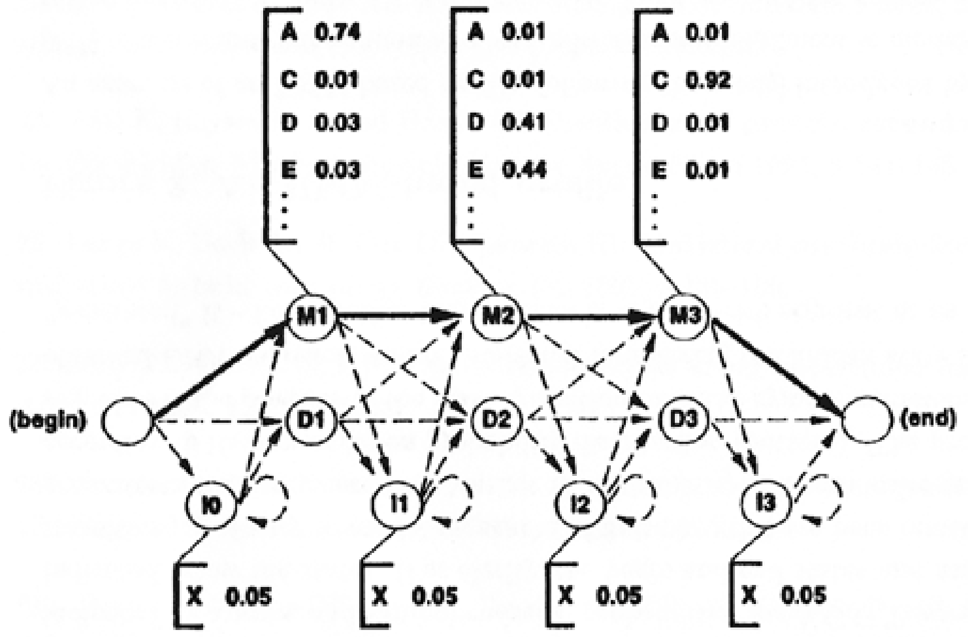
Similarities Between Hidden Markov Models and Turing Machines, and Possible Applications Towards Bioinformatics

Tyler Cheung

Over the past five or six years, Hidden Markov Models (HMMs) have become an important tool for certain bioinformatical tasks, including sequence and structural alignment. The advantage of an HMM is that it gives bioinformaticists, among other things, a rigorous mathematical method, through probability and combinatorics, of aligning two or more sequences of proteins and nucleic acids. However, mathematical constructs known as "Turing Machines" are very similar to how HMMs are defined, and thus, we wish to consider whether these machines have any bioinformatical applications.

HMMs can be briefly described as a series of states linked by several probabilistic paths. A typical model often used for sequence alignment is described in Krogh et al, 1994 (1). The states represent matches, gap insertions, or gap deletions, and the match states also define the probabilities for the occurrence of each amino acid. The arrangement of the states and the probabilities are often built by a given training set of amino acid sequences, i.e. a known set of orthologs. If one particular sequence in question wants to be scored against the known training set, every possible path through the HMM that gives that sequence is determined and the total probability, or the sum of the probabilities of all the paths, is calculated. This probability then serves as the basis of the score. A diagram of the different states and their interrelations is given in Eddy, 1996 (2):

Figure 1: Sample HMM for Sequence Alignment/Generation



In this case, the M states are the match states with the amino acid probabilities, the D states are the deletion states, and the I states are the insertion states. Each of the arrows denoting the paths also would be assigned a probability. This would be a typical HMM for a 2 or 3 amino acid sequence.

A Turing Machine is similar to this in that it also is a set of states connected by paths. The Turing Machine was devised by British mathematician Alan Turing as a way of dealing with the decidability problem, or "Entscheidungsproblem" (3). The mathematician Kurt Gödel had shown logically that certain mathematical problems were unsolvable, and Turing's invention was a practical proof of that, but one which also happened to be programmable and thus became the foundation of modern computer science (4).

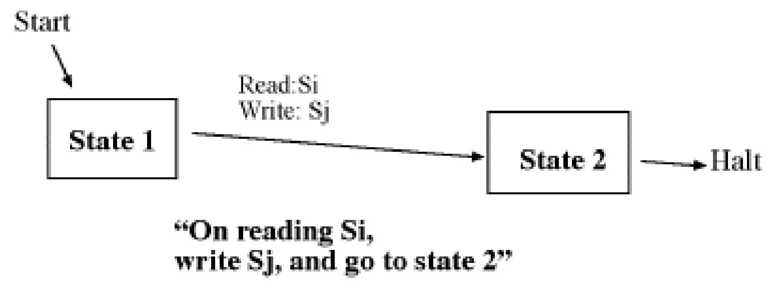
The machine can be loosely described as a "black box" equipped with a 1 dimensional data tape, which has three defined functions - read, write, and move. The box itself can assume any number of a finite number of states, similar to a Hidden Markov Model. A typical Turing machine moves in a series of time steps - each step involves reading from the tape, assuming the state based on the read input, moving along the tape, and writing. The tape itself contains an initial set of symbols, and to which are written new symbols by the Turing Machine. The machine itself moves one slot on the tape with each step.

We can basically define each Turing Machine with "quintuples," a set of five values, stating the read input, the current state, the new state, the direction to move left or right, and what to write. In mathematical terms, Feynman proposed representing the quintuples as a series of 5 variable sets, Q_i, S_i, Q_j, S_j, and D, subjected to the three functions F, G, and D where

$Q_j = F(Q_i, S_i)$
$S_j = G(Q_i, S_i)$
$D = D(Q_i, S_i)$

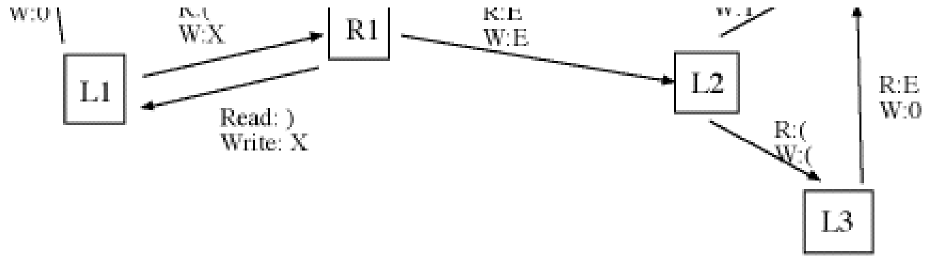
Q_i and S_i represent the initial states and input tape symbol, respectively, while Q_j and S_j represent the new state and the write symbol. D represents the direction for the machine to move (5). A more intuitive way of doing this is to draw a state diagram similar to that of a Hidden Markov Model in figure 1, as follows (5):

Fig. 2: Sample State Diagram for Turing Machine



A fairly good illustration of how a Turing Machine works is given by Richard Feynman in his series of lectures on computation at Caltech in the 1980's (5). A version of the machine he described is shown here. This particular machine checks for properly nested parentheses in a string of arbitrary length. The state diagram is as follows:

Fig. 3: State Diagram for Parenthesis Nestedness Checker



We supply a tape with the following symbols:

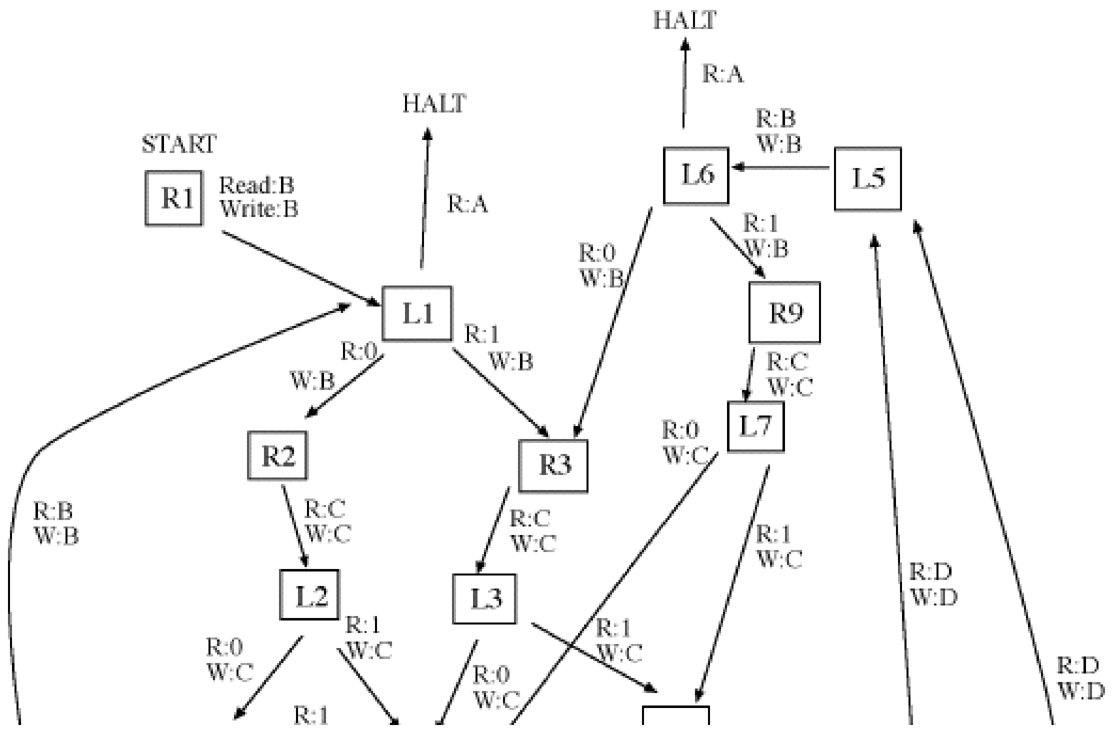
E	(	(	)	(	)	(	)	(	)	)	(	(	(	(	(	)	(	)	)	(	(	)	)	)	)	)	)	E	

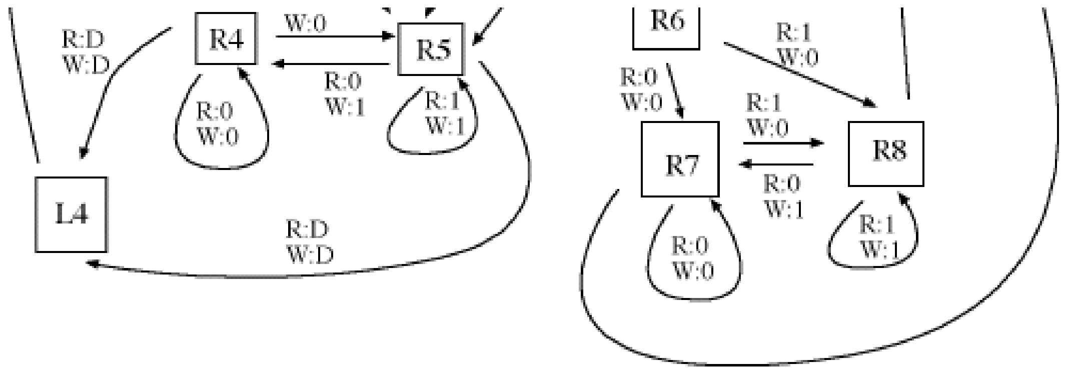
This is basically a parentheses string flanked by E symbols. We define the machine to start at the leftmost E, and work its way step by step to the right, keeping the machine in the state R1. It ignores any left parentheses, "(", i.e. writing left parentheses back into their original slots, until it hits a right parenthesis, ")", whereupon it replaces the ")" with an "X", switches into state L1 and changes direction back left. It replaces the first "(" it sees with another "X", switches back to R1 and changes direction back to the right. Eventually, it will reach either the left E or the right E, where we can tell whether the parentheses are properly nested by several of the following ways: if it has hit the left E, we know that the parentheses string is not properly nested, as this signifies an unpaired ")". We can replace that leftmost E with a "0" to signify "false". If it hits the right E, then we have the machine switch into state L2 where it moves left. If it hits an "X," nothing happens, but if it hits a "(," it switches into state L3. When it hits the left E, it writes a 1 if in state L2, and a 0 if in state L3. Thus, we can tell if the parentheses were properly nested by looking at the output at the leftmost E when the machine has finished. The finished state of the tape is thus:

0	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	)	)	E

Turing machines can also be used to define various mathematical functions. An example of a simple adder is given as follows:

Fig. 4: State Diagram for Turing Machine Adder





Initial tape:

A	1	1	0	1	B	1	0	0	1	C	D

The machine would move right until it hit a B, move left, and replace the digit immediately to the left with another B and change into a one of 2 states (R2, R3) representing the digit it just replaced, move right until it hit the C, and replace the digit to the left with a C, and assume one of three states representing the sum of that digit and the one represented by the corresponding digit of B (0, 1, 0 + carry - states R4, R5, and R6, respectively). It moves right, and writes that sum digit into the space directly to the right of the last C while shifting existing digits to the right of C further right by 1 slot. It then moves left to the leftmost B to deal with the next digits. If it needs to carry a digit, it assumes an alternate leftward state to signify this. Each new digit sum is inserted into the space directly to the right of the last C. We would, in theory, end up with the tape

A	B	B	B	B	B	C	C	C	C	C	1	0	1	1	0	D

The Turing Machine model is particularly powerful, as once the basic mathematical operations are defined, one can do a variety of tasks with them. In light of this, we can assume that Turing Machines can be applied to certain common bioinformatics tasks. One possible use would be to implement a simple Needleman-Wunsch scoring algorithm (6) by incorporating the adding machine described above. In comparing two sample protein sequences, say ALWE and ALPN, we could have a tape as follows:

X_0	A	L	W	E	X_1	A	L	P	N	X_2	X_3	X_4

A Turing Machine shuttling back and forth could easily match up the different pairs of amino acids, and write the rows, in sequential order, of a Needleman-Wunsch similarity matrix between X_2 and X_3. It could then sum up the similarity matrix and write the resulting rows between X_3 and X_4. If the rows are separated by a placeholder, you could even have the machine zip back and forth among the rows in X_3 and X_4, find the maxima of each row, and write down possible alignment paths through the Needleman-Wunsch matrix on the tape after X_4. Another application would be to simulate individual paths through a HMM, with probabilistic Turing Machines described in Leeuw et al. 1956 (7). Such machines have the functions F, G, or D of the quintuples set to generate a random values from the finite set of possible Turing machine states or outputs. We could set up a generic input tape to such a machine, such as

X_1	X_2	X_3	...	X_{20}

for a random 20 amino acid sequence from a given HMM configuration, minus any gaps that might be inserted, and run it through a Turing Machine with a state diagram and quintuple functions defined to be equivalent to the HMM. The 1 letter abbreviation for each amino acid, or a placeholder for gaps, could be inserted as the Turing machine steps over each X slot, thus generating on the tape a random sequence from the HMM. Or, we could have a given amino acid sequence on a tape, i.e.

T	Y	N	G	A	A	V	X1

and we could somehow have a Turing Machine defined with a HMM state diagram step through the sequence, evaluate all the possible paths through the state diagram, and calculate the sum of the probabilities, giving the score.

All this leads to the obvious question - what advantages do Turing Machines offer over current sequence alignment techniques? With the computers currently in use, a Turing Machine is a "virtual" or emulated implementation written on top of the existing native language of the computer. Thus, it is often a slower way of doing things rather than encoding programs with simple logic instructions with assembly language, or with languages such as C or perl. As noted above, however, their similarity with HMMs could mean that it would be easy to couple them with a HMM based alignment or profiling scheme. Also, the simularity of turing machines with natural biological objects that handle bits of information is striking - both usually operate on a 1-dimensional information storage system. What is a RNA polymerase but a Turing Machine defined as a copier with separate read and write tapes? What is a ribosome but a translating Turing Machine? The use of nucleic acid based Turing Machines have already been touched on in Adleman, 1994 (8), and Shapiro, 2000 (9). As the fields of biocomputing, protein design, protein engineering, and structure prediction improve, we may be able to design molecular Turing machines able to perform sequence alignments directly in the test tubes, or perhaps even in vivo sequence analysis or structure prediction. In other words, there may be a potential in biological systems for a native or "hard" Turing Machine.

Also, given the nature of the universal Turing Machine and the relatively free-form nature of the machine, there are probably a multitude of bioinformatics applications overlooked in this report and are just waiting to be discovered.

Works Cited

1. Krogh, A., Brown, B., Mian, I.S., Sjolander, K., Haussler, D. (1994). "Hidden Markov Models in Computational Biology: Applications to Protein Modeling." J. Mol. Biol. **235**:1501-1531.
2. Eddy, S.R. (1996). "Hidden Markov Models." Curr. Opin. Struc. Biol. **6**, 361-365.
3. Turing, A.M. "On the Computability of Numbers, With an Application to The Entscheidungsproblem." Proc. of the London Mathematical Society. **42**. 230-265. 1936.
4. Hopcraft, J.E. (1984). "Turing Machines." Sci. Am. **250**. 86-98.
5. Feynman, R.P. (1996). Feynman Lectures on Computation. Eds. T. Hey and R. W. Allen. New York: Addison-Wesley.
6. Needleman, S. B. and Wunsch, C. D. (1971). "A general method applicable to the search for similarities in the amino acid sequence of two proteins." J. Mol. Biol. **48**: 443-453.
7. Leeuw, K. de, Moore, E.F., Shannon, C.E., and Shapiro, N. (1956). "Computability by Probabilistic Machines." Annals of Mathematical Studies (Automata Studies). **34**: 183-212.
8. Adleman, L. (1994). "Molecular Computation of Solutions to Combinatorial Problems." *Science.* **266**: 1021-1024.
9. Shapiro, E. (2000). "A Mechanical Turing Machine: Blueprint for a Biomolecular Computer." <http://www.wisdom.weizmann.ac.il/~udi/DNA5/turing5.rtf> .