On the equivalence of probabilistic automata

Term Paper for CS-550

Abhinav Aggarwal
UNM ID : 101680103

Department of Computer Science
University of New Mexico

Spring 2015

Problem Statement

1. To understand the complexity and algorithms for comparing two probabilistic automata and determine, if they are not equal, the lexicographically minimum string on which they differ.
2. To study some metrics/techniques used for determining distance between two probabilistic automata.

Motivation

1. Question on equivalence of semantics of different program fragments in the homework.
2. The problem of equivalence of automaton relates directly to equivalence of finite state programs.
3. Studying metrics for computing distances between automata helps in understanding the efficiency of approximating the semantics of one program through another.

Probabilistic Automaton

A **probabilistic automaton** is a 5-tuple $\mathcal{A}(Q, \Sigma, M, \rho, \eta)$ where

1. Q is a finite set of states
2. Σ is the finite alphabet
3. $M : Q \times \Sigma \times Q \to [0, 1]$ is the transition function for the automaton. We interpret $M(q_1, x, q_2) = p$ as "*the probability that the state* q_1 *will change to* q_2 *upon reading* x *is* p".
4. $\rho : Q \to [0, 1]$ is the initial state distribution.
5. $\eta : Q \to [0, 1]$ is the acceptance probability for each state. We call $f \in Q$ an accepting state if $\eta(f) > 0$. The set of all accepting states is denoted F.

Equivalence Problem : Two probabilistic automata $\mathcal{A}_1$ and $\mathcal{A}_2$ are said to be equivalent for all $w \in \Sigma^*$, the probability of acceptance of x by $\mathcal{A}_1$ is the same as that by $\mathcal{A}_2$.

Probabilistic Automaton

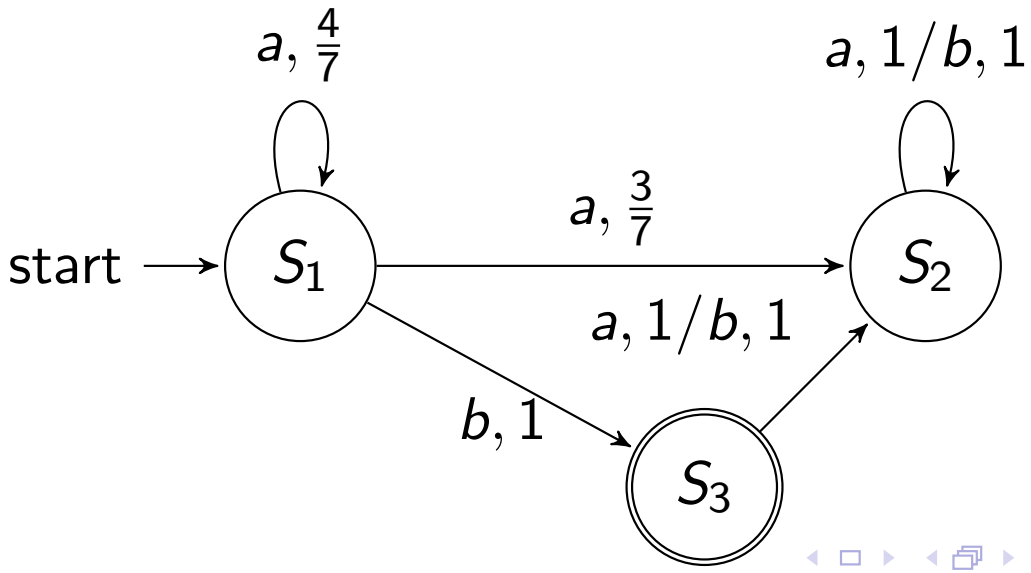
Facts and Notations :

1. The probability that string w is accepted is denoted $\mathcal{A}(w)$. The language of the automaton is thus $L(\mathcal{A}) = \{w \mid \mathcal{A}(w) > 0\}$.
2. Sum of output probabilities is 1 for every state : $\forall q_1 \in Q, x \in \Sigma \quad \sum_{q \in Q} M(q_1, x, q) = 1$
3. Every string begins with exactly one initial state: $\sum_{q \in Q} \rho(q) = 1$.
4. Denote the transition matrix for a given $x \in \Sigma$ as $T_x \in [0,1]^{Q \times Q}$, where $T_x(i,j) = M(i,x,j)$. This makes T_x a stochastic matrix for every x. We can extend this recursively to any string $w \in \Sigma^+$ as $T_{wx} = T_w T_x$, where $x \in Sigma$.
5. Interpret ρ as being a row vector $[0,1]^{1 \times Q}$ where $\rho(1,i) = \rho(i)$.
6. Interpret η as a column vector $[0,1]^{Q \times 1}$, where $\eta(i,1) = \eta(i)$.
7. The probability that the string $w \in \Sigma^+$ is accepted by $\mathcal{A}$ is thus given as $\mathcal{A}(w) = \rho T_w \eta$. Thus, we have $\sum_{w \in \Sigma^*} \mathcal{A}(w) = \sum_{w \in \Sigma^*} \rho T_w \eta = 1$
8. As a convention, the probability of accepting the empty string is given by $\rho\eta$.

Example

```
x:= " "
p:= unif(0,0.7)
while(p > 0.3)
  x:=append(x,'a')
  p:=unif(0,0.7)
x:=append(x,'b')
output x
```

Note : The output is $x = a^n b$ with probability $(\frac{4}{7})^n(\frac{3}{7})$ for $n \geq 0$.



Complexity of Language Equivalence

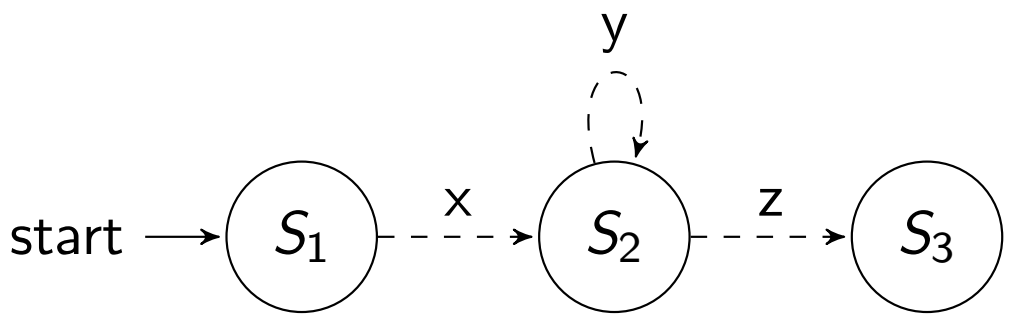
1. For *DFA*, we have *LINEAR* time by Hopcraft and Karp (1971).
2. For *NFA*, we have *PSPACE* − *HARD* by Meyer and Stockmeyer (1972).
3. For *PA*, we have *POLY* time by Tzeng (1992).
4. Natural Inclusion of *PA* : ($\mathcal{A}$ accepts each word with probability atleast as great as $\mathcal{B}$) : UNDECIDABLE even for fixed dimension automata (1989).
    1. Reduction of the problem NET : **N**on-**E**mptiness with **T**hreshold into it (Does there exist an $w \in \Sigma^*$ such that $\mathcal{A}(w) > \lambda$ for a given $0 \leq w \leq 1$)
    2. NET introduced by Rabin (1963) and proved undecidable by Paz (1971).
    3. Most elementary proof of undecidablility given by Blondel and Canterini (2003) based on the Post Correspondence Problem.
5. Approximate probabilistic equivalence - *UNDECIDABLE*

APEX

1. A tool for automatic verification of probabilistic programs.
2. Translates probabilistic programs into automata-theoretic representations using their **game semantics** - *EXPTIME* decidability.
3. Only externally observable computational steps are visible in the corresponding automata (internal local variable manipulation remains hidden)

An important fact about equivalence

If there exists a word $w \in \Sigma^*$ such that $\mathcal{A}(w) \neq \mathcal{B}(w)$, then there exists a word w' of length at most n, the number of states in $\mathcal{A}$, such that $\mathcal{A}(w') \neq \mathcal{B}(w')$.



Here $w = xy^m z$ for some $m \geq 1$ such that $|w| > n$. If $S_3 \notin F$, then $w' = xz \notin F$. Note that $|w'| \geq n$.

Tzeng's algorithm

1. A polytime algorithm for comparing two probabilistic automata over alphabet Σ having n_1 and n_2 states respectively : Runtime $= \mathcal{O}(|\Sigma|(n_1 + n_2)^4)$.
2. Idea taken from Schützenberger's minimization algorithm (1961) - $\mathcal{O}(|\Sigma|(n_1 + n_2)^3)$
3. In case of inequivalence, the algorithm outputs in polynomial time the lexicographically minimum string which the two automata accept with different probabilities.
4. Based on techniques from linear algebra.
5. Can be used to solve some more interesting problems in *POLY* time :
    1. Path equivalence for *NFA* without ε-transitions. (Already shown to be *POLY* time by Hunt and Stearns in 1986)
    2. δ-equivalence problem for probabilistic algorithms (Check if maximum difference in accepting probabilities for any word is atmost δ) - Non-terminating when δ is more than the theoretical maximum difference in accepting probabilities for any word.
    3. Language equivalence problem for unambiguous finite automata.

Tzeng's algorithm

Given two probabilistic automata $\mathcal{A}_1(S_1, \Sigma_1, M_1, \rho_1, \eta_1)$ and $\mathcal{A}_2(S_2, \Sigma_2, M_2, \rho_2, \eta_2)$ with number of states n_1 and n_2 respectively

1. Define for each string $x \in \Sigma^*$, the matrix

$$M_{\mathcal{A}_1 \oplus \mathcal{A}_2}(x) = \begin{bmatrix} T_{1,x} & 0_{n_1 \times n_2} \\ 0_{n_2 \times n_1} & T_{2,x} \end{bmatrix}$$

   Note that for any $\sigma \in \Sigma$, we have $M_{\mathcal{A}_1 \oplus \mathcal{A}_2}(x\sigma) = M_{\mathcal{A}_1, \oplus \mathcal{A}_2}(x) \times M_{\mathcal{A}_1 \oplus \mathcal{A}_2}(\sigma)$.
2. Also define for each string $x \in \Sigma^*$, the vector $P_{\mathcal{A}_1 \oplus \mathcal{A}_2}(x) = [\rho_1, \rho_2] M_{\mathcal{A}_1 \oplus \mathcal{A}_2}(x)$.
3. For the two automata, then define $H(\mathcal{A}_1, \mathcal{A}_2) = \{P_{\mathcal{A}_1 \oplus \mathcal{A}_2}(x) \mid x \in \Sigma^*\}$.
4. Then for equivalence we need $P_{\mathcal{A}_1 \oplus \mathcal{A}_2}(x)[\eta_1, -\eta_2] = 0$
5. **LEMMA :** If V is a basis for $H(\mathcal{A}_1, \mathcal{A}_2)$, then $\mathcal{A}_1$ and $\mathcal{A}_2$ are equivalent iff for all $v \in V$, we have $v \in \text{Null}[\eta_1, -\eta_2]$.

Tzeng's algorithm

1. Let T be a binary tree over words in Σ^*.
2. Define T_N as the tree formed by the nodes in $N \cup \{\text{node}(x\sigma) \mid \text{node}(x) \in N, \sigma \in \Sigma\}$

TABLE 1

Algorithm for equivalence of probabilistic automata.

Input: $U_1 = (S_1, \{0,1\}, M_1, \rho_1, F_1)$, $U_2 = (S_2, \{0,1\}, M_2, \rho_2, F_2)$;
1. Set V and N to be the empty set;
2. *queue* $\leftarrow$ *node*(λ);
3. **while** *queue* is not empty **do**
4. **begin** take an element *node*(x) from *queue*;
5. **if** $P_{U_1 \oplus U_2}(x) \notin span(V)$ **then**
6. **begin** add *node*$(x0)$ and *node*$(x1)$ to *queue*;
7. add vector $P_{U_1 \oplus U_2}(x)$ to V;
8. add *node*(x) to N
end;
end;
9. **if** $\forall v \in V$, $v[\eta_{F_1}, -\eta_{F_2}]^T = 0$ **then return**(*yes*)
10. **else return** (*lex*-min $\{x$: *node*$(x) \in N$, $P_{U_1 \oplus U_2}(x)[\eta_{F_1}, -\eta_{F_2}]^T \neq 0\}$)

Kullback-Leibler Divergence between PA

Given two *PA*, which are possibly not equivalent

- Aim **now** is "*approximate equivalence*"
- Need a measure of this approximation
- Information theoretic approach - "number of extra bits needed to deduce actual value from it's approximation"

Given two probabilistic automata $\mathcal{A}_1$ and $\mathcal{A}_2$, we have

$$KL(\mathcal{A}_1 \parallel \mathcal{A}_2) = \sum_{x \in \Sigma^*} \mathcal{A}_1(x) \log_2 \left(\frac{\mathcal{A}_1(x)}{\mathcal{A}_2(x)} \right)$$
$$\neq KL(\mathcal{A}_2 \parallel \mathcal{A}_1)$$

Summation over infinitely many words over the alphabet - the problem is proved $PSPACE - COMPLETE$ (2006) by Corinna Cortes, Mehryar Mohri and Ashish Rastogi.

FACT : Also proposed another measure - L_p distance - proved $NP - Hard$ for odd p by reduction from $MAX - CLIQUE$. For $p = 2$, the algorithm is $POLY$ time.