SIAM J. COMPUT.
Vol. 21, No. 2, pp. 216-227, April 1992

A POLYNOMIAL-TIME ALGORITHM FOR THE EQUIVALENCE OF PROBABILISTIC AUTOMATA*

WEN-GUEY TZENG†

Abstract. Two probabilistic automata are equivalent if for any string x, the two automata accept x with equal probability. This paper presents an $O((n_1+n_2)^4)$ algorithm for determining whether two probabilistic automata U_1 and U_2 are equivalent, where n_1 and n_2 are the number of states in U_1 and U_2, respectively. This improves the best previous result, which showed that the problem was in *coNP*.

The existence of this algorithm implies that the covering and equivalence problems for uninitiated probabilistic automata are also polynomial-time solvable. The algorithm used to determine the equivalence of probabilistic automata can also solve the path equivalence problem for nondeterministic finite automata without λ-transitions and the equivalence problem for unambiguous finite automata in polynomial time.

This paper studies the approximate equivalence (or δ-equivalence) problem for probabilistic automata. An algorithm for the approximate equivalence problem for positive probabilistic automata is given.

Key words. probabilistic automata, nondeterministic finite automata, unambiguous finite automata, equivalence, approximate equivalence, path equivalence

AMS(MOS) subject classifications. 68C25, 68D25

1. Introduction.

A probabilistic automaton is a finite state machine with probabilistic transitions among states. For each string x, a probabilistic automaton has a certain probability of accepting x. Two probabilistic automata are *equivalent* if for any string x the two automata accept x with equal probability. While the equivalence problem for probabilistic automata was known to be in *coNP* [7], the question of whether the equivalence problem was polynomial-time solvable was left open. In this paper we present a polynomial-time algorithm for this problem. Furthermore, if two probabilistic automata are not equivalent then we demonstrate that our algorithm will output in polynomial time the lexicographically minimum string which the two automata will accept with different probabilities. As a consequence, the covering and equivalence problems for uninitiated probabilistic automata are also polynomial-time solvable.

For a probabilistic automaton U, the *state distribution* induced by a string x is the vector of probabilities that U ends up in each state when the input of U is string x. Our approach to the problem makes use of an algorithm which finds a basis for the vector space generated by the state distributions induced by all strings.

The technique we use to solve the equivalence problem for probabilistic automata in polynomial time has an interesting application to the path equivalence problem for nondeterministic finite automata without λ-transitions (empty-string transitions). Two nondeterministic finite automata A_1 and A_2 are *path equivalent*, also called multiset equivalent in [4], if for each string x, the number of distinct accepting computation paths (or accepting state transition sequences) for x by A_1 is equal to that for x by A_2. The path equivalence problem for nondeterministic finite automata (with λ-transitions) is *PSPACE*-hard. Using difference equations, however, Hunt and Stearns were able to show that the path equivalence problem for nondeterministic finite automata without λ-transitions is solvable in polynomial time [4]. We give an alternative

* Received by the editors November 22, 1989; accepted for publication (in revised form) May 28, 1991. Part of this paper appeared in "The equivalence and learning of probabilistic automata" presented at the 30th Annual IEEE Symposium on Foundations of Computer Science, 1989, Research Triangle Park, North Carolina.

† Department of Computer Science, State University of New York at Stony Brook, Stony Brook, New York 11794. This research was supported in part by National Science Foundation grant CCR-8801575.

polynomial-time algorithm for this problem based on linear algebra. As a special case, the (language) equivalence problem for unambiguous finite automata is also polynomial-time solvable.

We also study the approximate equivalence (δ-equivalence) problem for probabilistic automata. Two probabilistic automata U_1 and U_2 are *δ-equivalent*, $\delta \geqq 0$, if for each string x the difference of its accepting probabilities by U_1 and U_2 is less than or equal to δ. A probabilistic automaton U is *positive* if for any two states q_1 and q_2 in U and any input symbol σ, the probability that U, on input σ, moves from state q_1 to state q_2 is greater than zero. We demonstrate an algorithm for the δ-equivalence problem for positive probabilistic automata. The algorithm will terminate except for the case where the input δ is equal to the maximum difference of accepting probabilities of a string.

2. Definitions. A (row) vector is *stochastic* if all its entries are greater than or equal to zero and sum to 1. A matrix is *stochastic* if all its row vectors are stochastic. Let $\mathcal{M}(i, j)$ be the set of all $(i \times j)$-dimensional stochastic matrices. Let λ be the empty string and $|x|$ be the length of string x. Let α^T be the transpose of the vector α. Let *span* be the function that maps a set of vectors to the vector space generated by the vectors in the set.

DEFINITION 2.1. A probabilistic automaton U is a 5-tuple (S, Σ, M, ρ, F), where $S = \{s_1, s_2, \cdots, s_n\}$ is a finite set of states, Σ is an input alphabet, M is a function from Σ into $\mathcal{M}(n, n)$, ρ is an n-dimensional stochastic row vector, and $F \subseteq S$ is a set of final states.

The vector ρ is called an *initial-state distribution* where the ith component of ρ indicates the probability of state s_i being the initial state. The value $M(\sigma)[i, j]$ is the probability that U moves from state s_i to state s_j after reading symbol $\sigma \in \Sigma$. We extend the domain of function M from Σ to Σ^* in the standard way, i.e., $M(x\sigma) = M(x)M(\sigma)$ for $x \in \Sigma^*$ and $\sigma \in \Sigma$, and $M(\lambda)$ is the $(n \times n)$-dimensional identity matrix. Let η_F be an n-dimensional row vector such that for $1 \leqq i \leqq n$,

$$\eta_F[i] = \begin{cases} 1 & \text{if } s_i \in F, \\ 0 & \text{otherwise.} \end{cases}$$

The *state distribution* induced by string x for U is

$$P_U(x) = \rho M(x),$$

where the ith component of $P_U(x)$ is the probability that U with initial-state distribution ρ moves to state s_i after reading x. We also define, for each string x,

$$Q_U(x) = M(x)(\eta_F)^T.$$

The *accepting probability* of x by U is

$$P_U(x)(\eta_F)^T = \rho Q_U(x),$$

which is the probability that U ends up in a final state when the input is string x.

DEFINITION 2.2. Let $U_1 = (S_1, \Sigma, M_1, \rho_1, F_1)$ and $U_2 = (S_2, \Sigma, M_2, \rho_2, F_2)$ be two probabilistic automata. Then U_1 and U_2 are said to be equivalent if for each string x, U_1 and U_2 accept x with equal probability, i.e., for all $x \in \Sigma^*$, $P_{U_1}(x)(\eta_{F_1})^T = P_{U_2}(x)(\eta_{F_2})^T$.

Let $U_1 = (S_1, \Sigma, M_1, \rho_1, F_1)$ and $U_2 = (S_2, \Sigma, M_2, \rho_2, F_2)$ be two probabilistic automata with number of states n_1 and n_2, respectively. We define, for each string x,

$$M_{U_1 \oplus U_2}(x) = \begin{bmatrix} M_1(x) & \mathbf{0}_{n_1 \times n_2} \\ \mathbf{0}_{n_2 \times n_1} & M_2(x) \end{bmatrix},$$

where $\mathbf{0}_{r\times s}$ is the $(r\times s)$-dimensional zero matrix; then we have, for any $\sigma\in\Sigma$,

$$M_{U_1\oplus U_2}(x\sigma)=M_{U_1\oplus U_2}(x)M_{U_1\oplus U_2}(\sigma).$$

We also define, for each string x,

$$\mathbf{P}_{U_1\oplus U_2}(x)=[\rho_1,\rho_2]M_{U_1\oplus U_2}(x).$$

Since the issue of computing with real numbers is subtle, in the rest of this paper we assume that all inputs consist of rational numbers and that each arithmetic operation on rational numbers can be done in constant time, unless stated otherwise.

3. Equivalence of probabilistic automata.

In this section we present a polynomial-time algorithm for the equivalence problem for probabilistic automata.

THEOREM 3.1. *There is an algorithm running in time $O((n_1+n_2)^4)$ that takes as input two probabilistic automata U_1 and U_2 and determines whether U_1 and U_2 are equivalent, where n_1 and n_2 are the number of states in U_1 and U_2, respectively. Furthermore, if U_1 and U_2 are not equivalent then the algorithm outputs the lexicographically minimum string which is accepted by U_1 and U_2 with different probabilities. This string will always be of length at most n_1+n_2-1.*

It has been shown [7] that two probabilistic automata are not equivalent if and only if there exists a string x of length at most n_1+n_2-1 such that $P_{U_1}(x)(\eta_{F_1})^T\neq P_{U_2}(x)(\eta_{F_2})^T$. This implies that the complexity of the equivalence problem for probabilistic automata is in *coNP*.

For two probabilistic automata U_1 and U_2, let

$$H(U_1,U_2)=\{P_{U_1\oplus U_2}(x)\colon x\in\Sigma^*\}.$$

Recall that U_1 and U_2 are equivalent if and only if

$$\forall x\in\Sigma^*,\quad P_{U_1}(x)(\eta_{F_1})^T=P_{U_2}(x)(\eta_{F_2})^T.$$

We can reformulate this equation as

$$\forall x\in\Sigma^*,\quad P_{U_1\oplus U_2}(x)[\eta_{F_1},-\eta_{F_2}]^T=0.$$

LEMMA 3.2. *Let U_1 and U_2 be two probabilistic automata. If V is a basis for span $(H(U_1,U_2))$ then U_1 and U_2 are equivalent if and only if for all $\mathbf{v}\in V$, $\mathbf{v}[\eta_{F_1},-\eta_{F_2}]^T=0$.*

Proof. Since the proof is straightforward, we omit it here. □

Because the dimension of the vector space $span(H(U_1,U_2))$ is at most n_1+n_2, the number of elements in V is at most n_1+n_2. A basic idea behind the design of our polynomial-time algorithm for the equivalence problem for probabilistic automata is to find a basis $V\subseteq H(U_1,U_2)$ for the vector space $span(H(U_1,U_2))$. If we are able to find such a basis in polynomial time then we can solve the equivalence problem for probabilistic automata in polynomial time.

Proof of Theorem 3.1. Without loss of generality, we let $\Sigma=\{0,1\}$. We define a binary tree T as follows. Tree T will have a node for every string in Σ^*. The root of T is $node(\lambda)$. Every $node(x)$ (where x is a string) in T has two children $node(x0)$ and $node(x1)$. Let $P_{U_1\oplus U_2}(x)$ be the (n_1+n_2)-dimensional vector associated with $node(x)$. For $node(x\sigma)$, $\sigma\in\Sigma$, its associated vector $P_{U_1\oplus U_2}(x\sigma)$ can be calculated by multiplying its parent's associated vector $P_{U_1\oplus U_2}(x)$ by $M_{U_1\oplus U_2}(\sigma)$.

The method we use to determine whether U_1 and U_2 are equivalent is to prune tree T. Initially, we set V to be the empty set. We then visit the nodes in T in *breadth-first* order. At each node $node(x)$, we verify whether its associated vector $P_{U_1\oplus U_2}(x)$ is linearly independent of V. If it is, we add the vector to V. Otherwise, we prune the subtree rooted at $node(x)$. We stop traversing tree T when every node in T is either visited or pruned. The vectors in the resulting set V will be linearly independent. We will show in Lemma 3.3 that the vectors in V form a basis for $span(H(U_1, U_2))$. The traversal order does not affect whether V is a basis for $span(H(U_1, U_2))$. A different traversal order will simply generate a different basis set. When U_1 and U_2 are not equivalent, a breadth-first traversal is necessary for finding the lexicographically minimum string whose accepting probabilities by U_1 and U_2 are different.

Our tree pruning algorithm appears in Table 1. In the algorithm, *queue* is a queue and N is a set of nodes. At the end of the algorithm, we verify whether $span(V)$ is a null space of linear transformation $[\eta_{F_1}, -\eta_{F_2}]^T$. If it is, then U_1 and U_2 are equivalent. Otherwise, we return the lexicographically minimum string in the set $\{x: node(x) \in N\}$ which is accepted by U_1 and U_2 with different probabilities.

Correctness. Let T_N be the tree formed by the nodes in

$$N \cup \{node(x\sigma): node(x) \in N, \sigma \in \Sigma\}$$

(the set of nodes that have been visited). Because the vectors in V are $(n_1 + n_2)$-dimensional, T_N has at most $n_1 + n_2$ internal nodes (those in N) and at most $n_1 + n_2 + 1$ leaves. Set V consists of the vectors associated with the internal nodes of T_N. Since we prune tree T at $node(x)$ when $P_{U_1 \oplus U_2}(x) \in span(V)$, the vectors associated with the leaves of T_N will be linearly dependent to the vectors in V. For example, in Fig. 1, the associated vectors $\mathbf{v}_1, \mathbf{v}_2, \cdots, \mathbf{v}_6$ of the internal nodes are linearly independent and the associated vectors of the leaves marked by $\otimes$ are linearly dependent to $\mathbf{v}_1, \mathbf{v}_2, \cdots, \mathbf{v}_6$.

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.    $queue \leftarrow node(\lambda)$;
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 $\forall \mathbf{v} \in V$, $\mathbf{v}[\eta_{F_1}, -\eta_{F_2}]^T = 0$ then return($yes$)
10.   else return ($lex$-min $\{x: node(x) \in N, P_{U_1 \oplus U_2}(x)[\eta_{F_1}, -\eta_{F_2}]^T \neq 0\}$);
```

We will prove that the vectors in the resulting set V form a basis for the vector space $span(H(U_1, U_2))$. For $i \geqq 0$, let

$$V_i = \{P_{U_1 \oplus U_2}(xy): node(x) \text{ is a leaf}, |y| = i\}.$$

Set V_0 is the set of vectors associated with the leaves of T_N and set V_i, $i \geqq 1$, is the set of vectors associated with the unvisited nodes of T which are of distance i from a leaf. It can be seen that

$$span\left(V \cup \bigcup_{i=0}^{\infty} V_i\right) = span(\{P_{U_1 \oplus U_1}(x): x \in \Sigma^*\}) = span(H(U_1, U_2)).$$

LEMMA 3.3. *For all $i \geqq 0$, $V_i \subseteq span(V)$.*

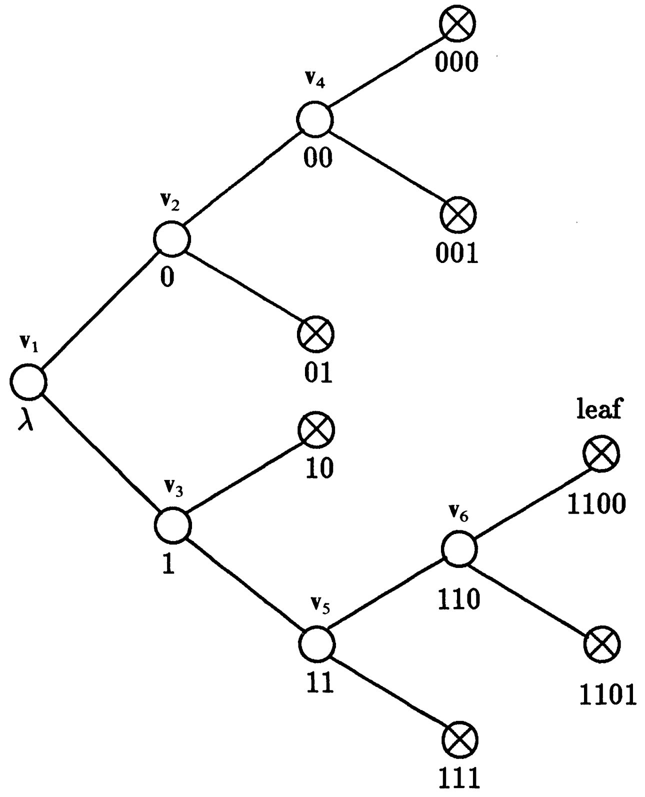

FIG. 1. *A T_N tree.*

Proof. Let $V=\{v_1, v_2, \cdots, v_r\}$ for some $r \leqq n_1+n_2$. We prove this lemma by induction on i. The base case $V_0 \subseteq \{vM_{U_1\oplus U_2}(\sigma): v \in V, \sigma \in \Sigma\} \subseteq span(V)$ follows from the algorithm. Assume that $V_i \subseteq span(V)$. Then for any x, y, and $\sigma \in \Sigma$ such that $node(x)$ is a leaf and $|y|=i$ it will be the case that

$$P_{U_1\oplus U_2}(xy\sigma)=P_{U_1\oplus U_2}(xy)M_{U_1\oplus U_2}(\sigma)=\left(\sum_{i=1}^{r} m_i v_i\right)M_{U_1\oplus U_2}(\sigma)$$

$$=\sum_{i=1}^{r} m_i(v_i M_{U_1\oplus U_2}(\sigma)) \in span(V \cup V_0)=span(V). \qquad \square$$

Thus the vectors in V form a basis for $span(H(U_1, U_2))$. By Lemma 3.2, the algorithm can determine whether U_1 and U_2 are equivalent by testing whether for all $v \in V$, $v[\eta_{F_1}, -\eta_{F_2}]^T=0$ holds.

LEMMA 3.4. *Let U_1 and U_2 be two probabilistic automata having number of states n_1 and n_2, respectively. If U_1 and U_2 are not equivalent then the algorithm in Table* 1 *outputs the lexicographically minimum string which is accepted by U_1 and U_2 with different probabilities. This string will be of length at most n_1+n_2-1.*

Proof. Let lex be the function that maps a string to its lexicographic order in Σ^*. Let x be the string returned by our algorithm. Assume that the lemma is false. Let y be the lexicographically minimum string such that $lex(y)<lex(x)$ and such that $P_{U_1\oplus U_2}(y)[\eta_{F_1}, -\eta_{F_2}]^T \neq 0$. Since $node(y) \notin N$, there must exist a leaf $node(z)$ such that $y=zw$ for some $w \in \Sigma^*$. Since we used a breadth-first traversal in our tree pruning

algorithm, the associated vector $P_{U_1\oplus U_2}(z)$ of *node*(z) will be in *span*$(\{P_{U_1\oplus U_2}(u)\colon u\in \Sigma^*,\ lex(u)<lex(z)\})$. Hence it will be the case that

$$\begin{aligned}P_{U_1\oplus U_2}(y)[\eta_{F_1},-\eta_{F_2}]^T &= P_{U_1\oplus U_2}(z)M_{U_1\oplus U_2}(w)[\eta_{F_1},-\eta_{F_2}]^T\\ &= \sum_{lex(u)<lex(z)} m_u P_{U_1\oplus U_2}(u)M_{U_1\oplus U_2}(w)[\eta_{F_1},-\eta_{F_2}]^T \qquad (1)\\ &= \sum_{lex(u)<lex(z)} m_u P_{U_1\oplus U_2}(uw)[\eta_{F_1},-\eta_{F_2}]^T.\end{aligned}$$

Because $lex(uw)<lex(zw)=lex(y)$ for any $lex(u)<lex(z)$, the value of equation (1) is zero. This contradicts the assumption $P_{U_1\oplus U_2}(y)[\eta_{F_1},-\eta_{F_2}]^T\neq 0$. Therefore the string x returned by our algorithm will be the lexicographically minimum string whose accepting probabilities by U_1 and U_2 are different. Furthermore, since no node in N is labeled by a string of length $>n_1+n_2-1$, the length of string x will be at most n_1+n_2-1. □

Complexity. Recall that we assume an arithmetic operation on rational numbers can be done in constant time. The binary tree T_N has at most n_1+n_2 internal nodes and thus at most n_1+n_2+1 leaves. The vector associated with *node*$(x\sigma)$ is calculated by multiplying its parent's associated vector $P_{U_1\oplus U_2}(x)$ by $M_{U_1\oplus U_2}(\sigma)$, which can be done in time $O((n_1+n_2)^2)$. To verify whether a set of (n_1+n_2)-dimensional vectors is linearly independent needs time $O((n_1+n_2)^3)$ [1]. Thus the total runtime is $O((n_1+n_2)^4)$. This completes the proof of Theorem 3.1. □

4. Equivalence of uninitiated probabilistic automata.

In this section we show that the covering and equivalence problems for uninitiated probabilistic automata are also polynomial-time solvable.

DEFINITION 4.1. An uninitiated probabilistic automaton U is a 4-tuple (S, Σ, M, F), where S is a finite set of n states, Σ is an input alphabet, M is a function from Σ into $\mathcal{M}(n, n)$, and $F\subseteq S$ is a set of final states.

Given an initial-state distribution ρ, we can form a probabilistic automaton $U(\rho)$ out of uninitiated automaton U.

DEFINITION 4.2. Let U_1 and U_2 be two uninitiated probabilistic automata. Then U_1 is said to cover U_2 if for any initial-state distribution ρ_2 for U_2 there is an initial-state distribution ρ_1 for U_1 such that $U_1(\rho_1)$ and $U_2(\rho_2)$ are equivalent.

DEFINITION 4.3. Let U_1 and U_2 be two uninitiated probabilistic automata. Then U_1 and U_2 are said to be equivalent if U_1 covers U_2 and U_2 covers U_1.

Let $U_1=(S_1,\Sigma,M_1,F_1)$ and $U_2=(S_2,\Sigma,M_2,F_2)$ be two uninitiated probabilistic automata having n_1 and n_2 states, respectively. Recall that $Q_{U_1}(x)=M_1(x)(\eta_{F_1})^T$. Let J be an $(n_1\times r)$-dimensional matrix whose column vectors $Q_{U_1}(x_1)$, $Q_{U_1}(x_2),\cdots,Q_{U_1}(x_r)$ for some $r\leqq n_1$ form a basis for the vector space *span*$(\{Q_{U_1}(x)\colon x\in\Sigma^*\})$. Let G be the $(n_2\times r)$-dimensional matrix whose column vectors are $Q_{U_2}(x_1)$, $Q_{U_2}(x_2),\cdots,Q_{U_2}(x_r)$. It has been shown [7] that if B is an $(n_2\times n_1)$-dimensional stochastic matrix such that $BJ=G$ then U_1 covers U_2 if and only if the condition for all $\sigma\in\Sigma$, $BM_1(\sigma)J=M_2(\sigma)G$ holds. Stochastic matrix B is a transformation from initial-state distributions for U_2 to those for U_1. The condition that for all $\sigma\in\Sigma$, $BM_1(\sigma)J=M_2(\sigma)G$ guarantees that $U_2(\rho_2)$ and $U_1(\rho_2B)$ are equivalent for any initial-state distribution ρ_2 for U_2. Since ρ_2 and B are stochastic, so is ρ_2B.

LEMMA 4.4 [7]. *Let U_1 and U_2 be two uninitiated probabilistic automata and let J and G be defined as above. Then U_1 covers U_2 if and only if there exists an $(n_2\times n_1)$-dimensional stochastic matrix B satisfying $BJ=G$ and if for any such B the condition $BM_1(\sigma)J=M_2(\sigma)G$ holds for all $\sigma\in\Sigma$.*

The problem of finding a stochastic matrix B such that $BJ = G$ can be reduced to the linear programming problem because of the "stochastic" restriction on B. If no such B exists then U_1 does not cover U_2. Once B has been found, it is easy to verify the condition. Thus our result implies the following.

THEOREM 4.5. *There is a polynomial-time algorithm that takes as input two uninitiated probabilistic automata U_1 and U_2 and determines whether U_1 covers U_2.*

Proof. By Lemma 4.4, we need to find matrices J, G, and B and then verify the condition for all $\sigma \in \Sigma$, $BM_1(\sigma)J = M_2(\sigma)G$. Using the idea that was used in the algorithm in Table 1, we can find in polynomial time strings $x_1, x_2, \cdots, x_r$ for some $r \leqq n_1$ such that $Q_{U_1}(x_1), Q_{U_1}(x_2), \cdots, Q_{U_1}(x_r)$ form a basis for $span(\{Q_{U_1}(x): x \in \Sigma^*\})$. Then matrices J and G can be calculated easily. The problem of finding a stochastic matrix B such that $BJ = G$ can be reduced to the linear programming problem, which is well known to be solvable in polynomial time [5]-[6]. If B does not exist, then U_1 does not cover U_2. Otherwise, we need to verify whether the condition for all $\sigma \in \Sigma$, $BM_1(\sigma)J = M_2(\sigma)G$ holds. If it does, then U_1 covers U_2. Otherwise, U_1 does not cover U_2. □

THEOREM 4.6. *There is a polynomial-time algorithm that takes as input two uninitiated probabilistic automata U_1 and U_2 and determines whether U_1 and U_2 are equivalent.*

Proof. By the definition of equivalence for uninitiated probabilistic automata, we need to verify that U_1 covers U_2 and that U_2 covers U_1. Theorem 4.5 implies that these tasks can be done in polynomial time. □

5. Approximate equivalence. In this section we consider the problem of determining whether two probabilistic automata are δ-equivalent. For $\delta \geqq 0$, two probabilistic automata are δ-equivalent if for every string x the probabilities that the two automata accept x differ by at most δ. Two equivalent probabilistic automata are zero-equivalent and any two probabilistic automata are 1-equivalent.

DEFINITION 5.1. Let $U_1 = (S_1, \Sigma, M_1, \rho_1, F_1)$ and $U_2 = (S_2, \Sigma, M_2, \rho_2, F_2)$ be two probabilistic automata and $\delta \geqq 0$ be a real number. Then U_1 and U_2 are said to be δ-equivalent if for all $x \in \Sigma^*$, $|P_{U_1}(x)(\eta_{F_1})^T - P_{U_2}(x)(\eta_{F_2})^T| \leqq \delta$.

We do not know the precise complexity for the δ-equivalence problem in general. In the following we consider the δ-equivalence problem for positive probabilistic automata only. A matrix is *positive* if all its entries are greater than zero. A probabilistic automaton $U = (S, \Sigma, M, \rho, F)$ is *positive* if all its transition matrices $M(\sigma)$, $\sigma \in \Sigma$, are positive. The reason for the restriction to positive matrices is that under this restriction the accepting probabilities of long strings can be estimated using the accepting probabilities of short strings [8].

In order to explain how we can estimate the accepting probabilities of long strings using short strings we will need an additional definition and lemma. For two probabilistic automata $U_1 = (S_1, \Sigma, M_1, \rho_1, F_1)$ and $U_2 = (S_2, \Sigma, M_2, \rho_2, F_2)$, we define

$$\delta^*(U_1, U_2) = \sup \{|P_{U_1}(\eta_{F_1})^T - P_{U_2}(\eta_{F_2})^T|: x \in \Sigma^*\}$$

and

$$\Delta(U_1, U_2) = \min_{i,j,k,\sigma} \{M_i(\sigma)[j, k]\}.$$

For a row vector $\alpha = [\alpha_1, \sigma_2, \cdots, \alpha_n]$, we define

$$|\sigma| = \max_{1 \leqq i \leqq n} \{\alpha_i\} \text{ and } \|\alpha\| = \max_{1 \leqq j,k \leqq n} \{|\alpha_j - \alpha_k|\}.$$

LEMMA 5.2 [8]. *Let α be a row vector, let A be a stochastic matrix, and let τ be $\min_{i,j} \{A[i, j]\}$. Then $|A\alpha^T - \alpha^T| \leqq \|\alpha\|$ and $\|A\alpha^T\| \leqq (1 - 2\tau)\|\alpha\|$.*

Let U_1 and U_2 be two positive probabilistic automata. We will show how to estimate the value $\delta^*(U_1, U_2)$. For any $0<\varepsilon\leqq 1$, let $m=O(\log(2/\varepsilon)/\Delta(U_1, U_2))$ such that $(1-2\Delta(U_1, U_2))^m\leqq\varepsilon/2$. For any string $x=yz$ such that $|x|\geqq m$ and $|z|=m$ it will be the case that

$$\begin{aligned}
&|\rho_1 Q_{U_1}(x)-\rho_2 Q_{U_2}(x)|\\
&\quad\leqq|\rho_1 Q_{U_1}(x)-\rho_1 Q_{U_1}(z)|+|\rho_2 Q_{U_2}(x)-\rho_2 Q_{U_2}(z)|\\
&\qquad+|\rho_1 Q_{U_1}(z)-\rho_2 Q_{U_2}(z)|\\
&\quad\leqq|Q_{U_1}(x)-Q_{U_1}(z)|+|Q_{U_2}(x)-Q_{U_2}(z)|+\delta'\\
&\quad=|M_1(y)Q_{U_1}(z)-Q_{U_1}(z)|+|M_2(y)Q_{U_2}(z)-Q_{U_2}(z)|+\delta'\\
&\quad\leqq\|Q_{U_1}(z)\|+\|Q_{U_2}(z)\|+\delta'\\
&\quad\leqq(1-2\Delta(U_1, U_2))^m\|\eta_{F_1}\|+(1-2\Delta(U_1, U_2))^m\|\eta_{F_2}\|+\delta'\\
&\quad\leqq\varepsilon/2+\varepsilon/2+\delta'\\
&\quad=\varepsilon+\delta'
\end{aligned}$$

where $\delta'=\max\{|\rho_1 Q_{U_1}(w)-\rho_2 Q_{U_2}(w)|: |w|\leqq m\}$. Since $\delta'\leqq\delta^*(U_1, U_2)$, we have $|\delta'-\delta^*(U_1, U_2)|\leqq\varepsilon$. Thus for any arbitrarily small value $\varepsilon>0$ there is an m large enough such that the difference between $\delta'=\max\{|\rho_1 Q_{U_1}(x)-\rho_2 Q_{U_2}(x)|: |x|\leqq m\}$ and $\delta^*(U_1, U_2)$ is $\leqq\varepsilon$.

We consider the following two questions.

Question 1. Given two positive probabilistic automata U_1 and U_2 and a number $0<\varepsilon\leqq 1$, find a number δ' such that $|\delta'-\delta^*(U_1, U_2)|\leqslant\varepsilon$.

Question 2. Given two positive probabilistic automata U_1 and U_2 and a number $0\leqq\delta\leqq 1$, determine whether U_1 and U_2 are δ-equivalent.

Our result for the first question is that for any two positive probabilistic automata U_1 and U_2 and for any $\varepsilon>0$, we can find in exponential time a δ' such that $|\delta'-\delta^*(U_1, U_2)|<\varepsilon$.

THEOREM 5.3. *There exists an algorithm that takes as input two positive probabilistic automata U_1 and U_2 together with a number $\varepsilon(0<\varepsilon\leqq 1)$ and outputs a number δ' such that $|\delta'-\delta^*(U_1, U_2)|\leqq\varepsilon$. In addition, the algorithm runs in time $O((2/\varepsilon)^{O(\log(k)/\Delta)})$ where Δ is the minimum entry of the transition matrices for U_1 and U_2 and k is the size of the input alphabet of U_1 and U_2.*

Proof. Let $(1-2\Delta)^m\leqq\varepsilon/2$. We evaluate $|\rho_1 Q_{U_1}(x)-\rho_2 Q_{U_2}(x)|$ for all strings x of length $\leqq m$ and let $\delta'=\max\{|\rho_1 Q_{U_1}(x)-\rho_2 Q_{U_2}(x)|: |x|\leqq m\}$. By the above discussion, we can see that $|\delta'-\delta^*(U_1, U_2)|\leqq\varepsilon$.

Since $m=O(\log(2/\varepsilon)/\Delta)$, we have to evaluate $k^m-1=(2/\varepsilon)^{O(\log(k)/\Delta)}$ values. For each string x such that $|x|\leqq m$, $\rho_1 Q_{U_1}(x)$ and $\rho_2 Q_{U_2}(x)$ can be evaluated in time $O(m\cdot n_1^2\cdot\log(2/\Delta)+m\cdot n_2^2\cdot\log(2/\Delta))$, where n_1 and n_2 are the number of states in U_1 and U_2, respectively. Therefore the total runtime is

$$O((2/\varepsilon)^{O(\log(k)/\Delta)}\cdot m\cdot(n_1^2+n_2^2)\cdot\log(2/\Delta))=O((2/\varepsilon)^{O(\log(k)/\Delta)}\cdot(n_1^2+n_2^2)).$$

Since we can assume that $n_1, n_2\leqq 1/\Delta$, the runtime can be simplified to

$$O((2/\varepsilon)^{O(\log(k)/\Delta)}).\qquad\Box$$

For the δ-equivalence problem for positive probabilistic automata (the second question), we first consider the case where $\delta\neq\delta^*(U_1, U_2)$. We let

$$N=\min\{n: |\delta-\delta^*(U_1, U_2)|\geqq 1/n,\ n \text{ is a natural number}\}$$

and $\delta' = \max\{|\rho_1 Q_{U_1}(x) - \rho_2 Q_{U_2}(x)| : |x| \leqq m\}$, where $(1-2\Delta(U_1, U_2))^m < 1/(2N)$. If $\delta' < \delta$, then

$$\delta^*(U_1, U_2) \leqq \delta' + 2(1-2\Delta(U_1, U_2))^m < \delta' + 1/N \leqq \delta.$$

If $\delta' > \delta$, then $\delta^*(U_1, U_2) > \delta$.

It seems difficult, however, to deal with the case where $\delta = \delta^*(U_1, U_2)$, except for the cases where $\delta = 1$ and $\delta = 0$ which can be solved by our previous algorithm. The main obstacle for the case where $\delta = \delta^*(U_1, U_2)$ is that the value $\delta^*(U_1, U_2)$ may occur in the limit in such a way that it may not be possible to calculate it in finite steps. Consider, for instance, the following example from [8]. Let $U' = (\{s_1, s_2\}, \{0, 1\}, M', [1, 0], \{s_2\})$, where

$$M'(0) = \begin{bmatrix} 1 & 0 \\ \frac{1}{2} & \frac{1}{2} \end{bmatrix} \quad \text{and} \quad M'(1) = \begin{bmatrix} \frac{1}{2} & \frac{1}{2} \\ 0 & 1 \end{bmatrix}.$$

It is easy to verify that for string $x = \sigma_1\sigma_2 \cdots \sigma_n$, $\sigma_i \in \Sigma$, $1 \leqq i \leqq n$, the accepting probability of x by U is

$$[1, 0]M'(x)[0, 1]^T = \frac{\sigma_n}{2} + \frac{\sigma_{n-1}}{2^2} + \cdots + \frac{\sigma_1}{2^n}.$$

Let U'' be the probabilistic automaton that accepts every string with probability $\frac{1}{3}$. Then we have $\delta^*(U', U'') = \frac{2}{3}$, which occurs when $x = 111 \cdots$.

Our result about the δ-equivalence problem for positive probabilistic automata is as follows.

THEOREM 5.4. *There exists an algorithm which takes as input two positive probabilistic automata U_1 and U_2 and a number $\delta(0 \leqq \delta \leqq 1)$ and determines whether U_1 and U_2 are δ-equivalent when $\delta \neq \delta^*(U_1, U_2)$. In addition, when the algorithm terminates, it runs in time $O((\max\{2, N\})^{O(\log(k)/\Delta)})$ where N is the minimum integer such that $|\delta - \delta^*(U_1, U_2)| \geqq 1/N$, Δ is the minimum entry of transition matrices for U_1 and U_2 and k is the size of the input alphabet of U_1 and U_2.*

Proof. Our algorithm appears in Table 2. When $\delta = \delta^*(U_1, U_2)$, the conditions in steps 6 and 7 will not hold and the algorithm will not terminate. The complexity analysis is similar to that of Theorem 5.3. □

6. Path equivalence of nondeterministic finite automata without λ-transitions.

In this section we apply our algorithm to the path equivalence problem for nondeterministic finite automata without λ-transitions. We first give definitions of nondeterministic finite automata, computation paths, and path equivalence.

TABLE 2
Algorithm for δ-equivalence of probabilistic automata.

```
Input: $U_1 = (S_1, \Sigma, M_1, \rho_1, F_1)$, $U_2 = (S_2, \Sigma, M_2, \rho_2, F_2)$, $0 \leqq \delta \leqq 1$;
1.  $\Delta \leftarrow \min_{\sigma,i,j,k}\{M_i(\sigma)[j, k]\}$;
2.  if $\delta = 1$ then return (yes);
3.  if $\delta = 0$ then run the algorithm in Table 1;
4.  $i \leftarrow 0$;
5.  while true do
6.  begin if $\exists x$, $|x| = i$, $|\rho_1 Q_{U_1}(x) - \rho_2 Q_{U_2}(x)| > \delta$ then return (no);
7.        if $\max\{|\rho_1 Q_{U_1}(x) - \rho_2 Q_{U_2}(x)| : |x| = i\} + 2(1-2\Delta)^i \leqq \delta$
          then return (yes);
8.        $i \leftarrow i + 1$;
9.  end;
```

DEFINITION 6.1. A nondeterministic finite automaton A is a 5-tuple $(S, \Sigma, \delta, s_1, F)$, where S is a finite set of states, Σ is an input alphabet, δ is a transition function from $S\times(\Sigma\cup\{\lambda\})$ into the power set of S, s_1 is the initial state, and $F\subseteq S$ is a set of final states.

DEFINITION 6.2. Let $A=(S, \Sigma, \delta, s_1, F)$ be a nondeterministic finite automaton. A computation path θ for A is a finite nonempty sequence

$$((q_1, a_1), (q_2, a_2), \cdots, (q_n, a_n), q_{n+1}),$$

where $(q_i, a_i)\in S\times(\Sigma\cup\{\lambda\})$ for $1\leqq i\leqq n$, $q_{n+1}\in S$, and $q_{i+1}\in\delta(q_i, a_i)$ for $1\leqq i\leqq n$. If $q_1=s_1$ and $q_{n+1}\in F$ then the sequence θ is said to be an accepting computation path for x, where $x=a_1a_2\cdots a_n$.

A nondeterministic finite automaton with λ-transitions could accept a finite string via an infinite number of computation paths. Consequently, the path equivalence (called multiset equivalence in [4]) of nondeterministic finite automata is defined as follows.

DEFINITION 6.3. Let A_1 and A_2 be two nondeterministic finite automata. Then A_1 and A_2 are said to be path equivalent if for each string x the number of distinct accepting computation paths for x by A_1 is equal to that for x by A_2 or both are infinite.

The path equivalence problem for nondeterministic finite automata (with λ-transitions) is *PSPACE*-hard. We do not know whether the problem is decidable.

THEOREM 6.4. *The following problem is PSPACE-hard: Given two nondeterministic finite automata (with λ-transitions) A_1 and A_2, determine whether A_1 and A_2 are path equivalent.*

Proof. We reduce the *PSPACE*-complete problem of determining whether a nondeterministic finite automaton accepts all strings [3] to this problem. For each nondeterministic finite automaton $A=(S, \Sigma, \delta, s_1, F)$, we construct the following two automata A_1 and A_2. Automaton A_1 is $(S, \Sigma, \delta', s_1, F)$, where $\delta'(s_1, \lambda)=s_1$ and $\delta'(q, a)=\delta(q, a)$ for any $q\in S$ and $a\in\Sigma\cup\{\lambda\}$. A string is accepted by A if and only if it is accepted by A_1 via an infinite number of computation paths. Automaton A_2 is $(\{s_1\}, \Sigma, \delta_2, s_1, \{s_1\})$, where $\delta_2(s_1, a)=s_1$ for any $d\in\Sigma\cup\{\lambda\}$, which accepts every string via an infinite number of computation paths. It is easy to see that A accepts all strings if and only if A_1 and A_2 are path equivalent. □

Therefore we only consider the path equivalence problem for nondeterministic finite automata without λ-transitions. Let $A=(S, \Sigma, \delta, s_1, F)$ be an n-state nondeterministic finite automaton without λ-transitions. The transition function δ can be represented as $(n\times n)$-dimensional transition matrices $M(\sigma)$, $\sigma\in\Sigma$, such that

$$M(\sigma)[i, j]=\begin{cases}1 & \text{if } s_j\in\delta(s_i, \sigma),\\ 0 & \text{otherwise.}\end{cases}$$

Note that the above transition matrices $M(\sigma)$ are not stochastic. For each string x, $P_A(x)\eta_F$, which is $[1, 0, 0, \cdots, 0]M(x)\eta_F$, is equal to the number of distinct accepting computation paths for x by A.

The following result about the path equivalence problem for nondeterministic finite automata without λ-transitions was first proved in [4] (cf. [9]); we give a different proof using our techniques.

THEOREM 6.5. *There is a polynomial-time algorithm that takes as input two nondeterministic finite automata A_1 and A_2 without λ-transitions and determines whether A_1 and A_2 are path equivalent. If A_1 and A_2 are not path equivalent then the algorithm outputs the lexicographically minimum string x which is accepted by A_1 and A_2 via different* *numbers of computation paths. The length of x will be less than the total number of states in A_1 and A_2.*

Proof. We use transition matrices to represent the transition functions of nondeterministic finite automata as described earlier and apply the algorithm in Table 1 to verify whether A_1 and A_2 are path equivalent. The rest of the proof is identical to that of Theorem 3.1. □

7. Equivalence of unambiguous finite automata.

A nondeterministic finite automaton A is of *finite degree of ambiguity* if there exists a constant k such that every string is accepted by A via at most k distinct computation paths. If k is 1 then the automaton is *unambiguous.*

In [9] Stearns and Hunt showed that, for any fixed k, there exists a polynomial-time algorithm for the (language) equivalence problem for nondeterministic finite automata of degree of ambiguity less than or equal to k. Our algorithm is not applicable to the problem in general. It can, however, solve the (language) equivalence problem for unambiguous finite automata in polynomial time, because two unambiguous finite automata are (language) equivalent if and only if they are path equivalent.

The result of Theorem 7.2 about the equivalence problem for unambiguous finite automata was first proved in [9]; we give a different proof using our techniques.

LEMMA 7.1 [9]. *There is a polynomial-time algorithm that takes as input an n-state nondeterministic finite automaton A with λ-transitions and outputs an equivalent nondeterministic finite automaton A' without λ-transitions and having at most n states. In addition, if A is of degree of ambiguity k then A' is of degree of ambiguity at most k.*

THEOREM 7.2. *There is a polynomial-time algorithm that takes as input two unambiguous finite automata A_1 and A_2 and determines whether A_1 and A_2 are equivalent. If A_1 and A_2 are not equivalent then the algorithm outputs the lexicographically minimum string which is accepted by A_1, but not by A_2, or vice versa. Furthermore, this string will be of length less than the total number of states in A_1 and A_2.*

Proof. By Lemma 7.1, we can transform A_1 and A_2 to their equivalent unambiguous finite automata A_1' and A_2' without λ-transitions, respectively. Although the transformation does not preserve the number of accepting computation paths for each string in the case of general nondeterministic finite automata, it does preserve the number of accepting computation paths for each string in the case of unambiguous finite automata. We then apply the algorithm in Table 1 to verify whether A_1' and A_2' are path equivalent. The rest of the proof is identical to that of Theorem 3.1. □

8. Conclusion.

In the complexity analysis of our algorithm for the equivalence problem for probabilistic automata, we assumed that each arithmetic operation on rational numbers could be done in constant time. This assumption, however, is not essential to the polynomial execution time of our algorithm. It is possible to show that the number of bits in the vectors associated with the nodes in tree T_N grows polynomially and that our algorithm will still run in polynomial time if arithmetic operations on rational numbers require time proportional to their number of bits.

A probabilistic automaton U with a real-valued cut-point π defines the language consisting of those strings accepted by U with probability greater than π. The equivalence problem for cut-point probabilistic automata is quite different from the (exact) equivalence problem for probabilistic automata because the emptiness problem for cut-point probabilistic automata is undecidable [7]. This undecidability result holds even if all input numbers are rational [2]. By a simple reduction, we can see that the equivalence problem for cut-point probabilistic automata is *undecidable* even if all input numbers are rational.

Acknowledgments. The author would like to thank Professor Ker-I Ko for his support and for many helpful discussions, Professor Scott Smolka, who introduced him to the problem of determining the equivalence of probabilistic automata, and the referees for their valuable comments.

REFERENCES

[1] D. K. FADDEEV AND V. N. FADDEEVA, *Computational Methods of Linear Algebra*, Freeman, San Francisco, 1963.

[2] S. GINSBURG, *The Mathematical Theory of Context-Free Languages*, McGraw-Hill, New York, 1966.

[3] J. HOPCROFT AND J. ULLMAN, *Introduction to Automata Theory, Languages and Computation*, Addison-Wesley, Reading, MA, 1979.

[4] H. B. HUNT III AND R. E. STEARNS, *On the complexity of equivalence, nonlinear algebra, and optimization on rings, semirings, and lattices*, extended abstract, Computer Science Department, State University of New York, Albany, NY, TR 86-23, 1986.

[5] N. KARMARKAR, *A new polynomial-time algorithm for linear programming*, Combinatorica, 4 (1984), pp. 373-395.

[6] L. G. KHACHIYAN, *A polynomial algorithm in linear programming*, Soviet Math. Dokl., 20 (1979), pp. 191-194.

[7] A. PAZ, *Introduction to Probabilistic Automata*, Academic Press, New York, 1971.

[8] M. O. RABIN, *Probabilistic automata*, Inform. Control, 6 (1963), pp. 230-245.

[9] R. E. STEARNS AND H. B. HUNT III, *On the equivalence and containment problems for unambiguous regular expressions, regular grammars, and finite automata*, SIAM J. Comput., 14 (1985), pp. 598-611.