International Journal of Foundations of Computer Science Vol. 13 No. 1 (2002) 83–97

AN EFFICIENT ALGORITHM FOR CONSTRUCTING MINIMAL COVER AUTOMATA FOR FINITE LANGUAGES

CEZAR CAMPEANU
University of Bucharest, Bucharest, Romania

ANDREI PAUN and SHENG YU
Department of Computer Science, University of Western Ontario
London, Ontario, Canada N6A 5B7
apaun@csd.uwo.ca
syu@csd.uwo.ca

Received 29 December 2000
Revised 20 April 2001
Communicated by Sheng Yu

ABSTRACT

The concept of cover automata for finite languages was formally introduced in [3]. Cover automata have been studied as an efficient representation of finite languages. In [3], an algorithm was given to transform a DFA that accepts a finite language to a minimal deterministic finite cover automaton (DFCA) with the time complexity $O(n^4)$, where n is the number of states of the given DFA. In this paper, we review the basic concept of cover automata and describe a new efficient transformation algorithm with the time complexity $O(n^2)$, which is a significant improvement from the previous algorithm.

Keywords: Finite languages, Deterministic Finite Automata, Cover Language, Deterministic Cover Automata.

1. Introduction

Finite languages are perhaps the most often used but the least studied family of languages in the formal language family hierarchy. In the sixties and seventies, although finite languages were studied, they appeared as fragmented pieces hidden in other topics and titles [10]. Only recently, several aspects of finite languages, e.g., the state complexity and decompositions, have been systematically studied [1, 2, 8, 9].

Finite languages have many practical applications [11, 4]. However, many finite languages used in applications need thousands or even millions of states if they are represented by deterministic finite automata (DFA) or similar structures. In [3], deterministic finite cover automata (DFCA) were introduced as an alternative representation of finite languages. Experiments have shown that, in many cases, DFCA are much smaller in size than their corresponding minimal DFA [9].

Let L be a finite language and l the length of the longest word(s) in L. Intuitively,

a DFCA A for L is a DFA that accepts all words in L and possibly additional words of length greater than l. So, a word w is in L if and only if it is accepted by A (as a DFA) and it has a length less than or equal to l. Note that checking the length of a word is usually not an extra burden in practice since the length of an input word is kept anyway in most applications.

In order to explain intuitively the notion of a DFCA, we give a very simple example in the following. Let $\Sigma = \{a, b, c\}$ be the alphabet and $L = \{abc, ababc, abababc\}$ a finite language over Σ. Clearly, the length of the longest word in L is 7, i.e., $l = 7$. The minimal DFA accepting L is shown in Figure 1, which has 8 states (9 if we count also the sink state). A minimal DFCA is shown in Figure 2, which has only 4 states (5 if we count also the sink state).

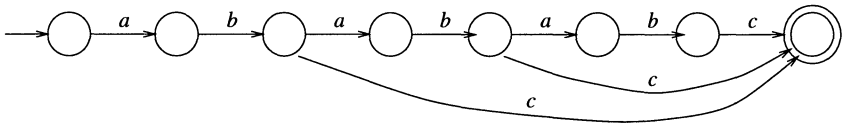

Fig. 1. The minimal DFA accepting L.

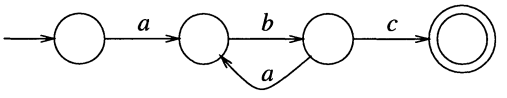

Fig. 2. The minimal DFCA for L with $l = 7$.

In [3], an algorithm was given for constructing a minimal DFCA from a given DFA that accepts a finite language. The time complexity of the algorithm is $O(n^4)$, where n is the number of states of the DFA. Note that the number of transitions of a DFA is linear to its number of states. A sketch of a two-step procedure for an essentially the same problem was given in [10] [on page 99]. This problem was also mentioned in [6] as a special case of Problem AL8. However, in [10], neither any detail nor an analysis of the algorithm was given. Under the best assumptions, their algorithm is no better than the $O(n^4)$ algorithm given in [3]. In this paper, we give an $O(n^2)$ algorithm for the construction of a minimal DFCA from a given DFA. The new algorithm is not only a significant improvement from the previous algorithm [3] in time complexity, but also much easier to comprehend and to implement. The new algorithm is based on several new results on the similarity relations on words and states, respecitively. So, in this paper, we also present those new results fundamental to the theory of cover automata.

The new algorithm differs from the algorithm given in [3] mainly at how to compute the similarity (or dissimilarity) relation between states. The new algorithm computes the pairs of states that are dissimilar and propagates the dissimilarity relations, rather than to compute directly the similarity relation as in the algorithm in [3]. A new algorithm is also given for merging similar states, which is simpler than the one given in [3]. Before giving the details of the new algorithms, we prove

several new theorems on the similarity relations which form the theoretical basis of the new algorithm.

In the next section, we give the basic definitions and notation, as well as the basic results, on cover languages and cover automata. In Section 3, we prove two theorems which are essential to the new algorithm. In Section 4, we describe our new algorithm and analyze its complexity. In the last section, we conclude the paper.

2. Preliminaries

First, we give the basic definitions and notation for cover languages, cover automata, and the similarity relation. Then we list some basic results, which are relevant to this paper, without giving any proofs. Detailed explanations and proofs can be found in [3] or [9].

Let S be a finite set of symbols and n a nonnegative integer. We define $S^n = \{w \in S^* \mid |w| = n\}$, where $|w|$ denote the number of appearances of symbols in w, i.e., the length of w. We also define $S^{\leq n} = \cup_{i=0}^{n} S^i$.

If $T = \{t_1, \ldots, t_k\}$ is an ordered set, $k > 0$, the lexicographic ordering on T^*, denoted $\prec$, is defined by: $x \prec y$ iff $|x| < |y|$ or $|x| = |y|$ and $x = zt_iv$, $y = zt_ju$, $i < j$, for some $z, u, v \in T^*$ and $1 \leq i, j \leq k$. Denote $x \preceq y$ if $x \prec y$ or $x = y$.

We say that x is a prefix of y, denoted $x \preceq_p y$, if $y = xz$ for some $z \in T^*$.

Definition 1 *Let $L \subset \Sigma^*$ be a finite language over an alphabet Σ and l the length of the longest word(s) in L. A language L' over Σ is called a cover language of L if $L' \cap \Sigma^{\leq l} = L$.*

Definition 2 *A cover automaton for a finite language L is a finite automaton A such that the language accepted by A, i.e., $L(A)$, is a cover language of L. If A is a DFA, then A is called a deterministic finite cover automaton (DFCA) for L.*

We often use the term cover automaton casually to mean DFCA in this paper.

In the following, we give the basic definitions regarding the similarity relation (on Σ^*), which is a generalization of the equivalence relation $\equiv_L$ ($x \equiv_L y$: $xz \in L$ iff $yz \in L$ for all $z \in \Sigma^*$).

We first define the similarity relation on words with respect to a finite language, and then the similarity relation on states of a DFA that accepts a finite language. The notion of similarity between words was first introduced in [7], and then studied in [5], [3], [9], etc. The concept of the similarity relation on words is the basis for the similarity relation on states of a DFA.

Definition 3 *Let L be a finite language over the alphabet Σ and l the length of the longest word(s) in L. Let $x, y \in \Sigma^*$. We define the following relation:*

- *$x \sim_L y$ if for all $z \in \Sigma^*$ such that $| xz | \leq l$ and $| yz | \leq l, xz \in L$ iff $yz \in L$;*

and we will write $x \not\sim_L y$ if $x \sim_L y$ does not hold.

The relation $\sim_L$ is called the *similarity* relation with respect to L. We will use $x \sim y$ instead of $x \sim_L y$ when L is clearly understood from the context. Note that the relation $\sim_L$ is reflexive, symmetric, but NOT transitive.

Lemma 1 *Let $L \subseteq \Sigma^*$ be a finite language and $x, y, z \in \Sigma^*$, $|x| \leq |y| \leq |z|$. The following statements hold:*

(1) If $x \sim_L y$, $x \sim_L z$, then $y \sim_L z$.

(2) If $x \sim_L y$, $y \sim_L z$, then $x \sim_L z$.

(3) If $x \sim_L y$, $y \not\sim_L z$, then $x \not\sim_L z$.

(4) If $x \sim_L y$, $x \not\sim_L z$, then $y \not\sim_L z$

So, in other words, if $|x| \leq |y| \leq |z|$ and $x \sim_L y$, then we have $x \sim_L z$ iff $y \sim_L z$

Proof. The first three statements were proved in [5], we now prove the last statement: Let $x \sim_L y$ and $x \not\sim_L z$ and assume that $y \sim_L z$. Then from statement (2) we get $x \sim_L z$, which is contrary to our assumption. So, we have $y \not\sim_L z$. □

If $x \not\sim_L y$ and $y \sim_L z$, we cannot say anything about the similarity relation between x and z. Similarly, if $x \not\sim_L y$ and $y \not\sim_L z$, we cannot conclude anything either between x and z.

Example: Let $L = \{aa, aaa, bbb, bbbb, aaab\}$. Consider the following four different cases:

(1) $x = aa$, $y = bbb$, $z = bbbb$

(2) $x = aa$, $y = bba$, $z = abba$

(3) $x = aa$, $y = bb$, $z = bbbb$

(4) $x = aa$, $y = bb$, $z = bbb$

In both the first two cases, we have $x \not\sim_L y$ and $y \sim_L z$; but we have $x \sim_L z$ in (1) and $x \not\sim_L z$ in (2). Similarly, in both (3) and (4) we have $x \not\sim_L y$ and $y \not\sim_L z$; but we have $x \sim_L z$ in (3) and $x \not\sim_L z$ in (4).

Definition 4 *Let $L \in \Sigma^*$ be a finite language.*

1. *A set $S \subseteq \Sigma^*$ is called an L-similarity set if $x \sim_L y$ for every pair $x, y \in S$.*
2. *A sequence of words $(x_1, \ldots, x_n)$ over Σ is called a dissimilar sequence of L if $x_i \not\sim_L x_j$ for each pair i, j, $1 \leq i, j \leq n$ and $i \neq j$.*
3. *A dissimilar sequence $(x_1, \ldots, x_n)$ is called a canonical dissimilar sequence of L if there exists a partition $\pi = \{S_1, \ldots, S_n\}$ of Σ^* such that for each i, $1 \leq i \leq n$, $x_i \in S_i$, and S_i is a L-similarity set.*
4. *A dissimilar sequence $(x_1, \ldots, x_n)$ of L is called a maximal dissimilar sequence of L if for any dissimilar sequence $(y_1, \ldots, y_m)$ of L, $m \leq n$.*

Theorem 1 *A dissimilar sequence of L is a canonical dissimilar sequence of L if and only if it is a maximal dissimilar sequence of L.*

Proof. Let L be a finite language. Let $[x_1, \ldots, x_n]$ be a canonical dissimilar sequence of L and $\pi = \{S_1, \ldots, S_n\}$ the corresponding partition of Σ^* such that for each i, $1 \leq i \leq n$, S_i is an L-similarity set. Let $[y_1, \ldots, y_m]$ be an arbitrary dissimilar sequence of L. Assume that $m > n$. Then there are y_i and y_j, $i \neq j$, such that $y_i, y_j \in S_k$ for some k, $1 \leq k \leq n$. Since S_k is a L-similarity set, $y_i \sim_L y_j$. This is a contradiction. Then, the assumption that $m > n$ is false, and we conclude that $[x_1, \ldots, x_n]$ is a maximal dissimilar sequence.

Conversely, let $[x_1, \ldots, x_n]$ a maximal dissimilar sequence of L. Without loss of generality we can suppose that $|x_1| \leq \ldots \leq |x_n|$. For $i = 1, \ldots, n$, define

$$X_i = \{y \in \Sigma^* \mid y \sim_L x_i \text{ and } y \notin X_j \text{ for } j < i\}.$$

Note that for each $y \in \Sigma^*$, $y \sim_L x_i$ for at least one i, $1 \leq i \leq n$, since $[x_1, \ldots, x_n]$ is a *maximal* dissimilar sequence. Thus, $\pi = \{X_1, \ldots, X_n\}$ is a partition of Σ^*. The remaining part of the proof is to show that each X_i, $1 \leq i \leq n$, is a similarity set.

We assume the contrary, i.e., for some i, $1 \leq i \leq n$, there exist $y, z \in X_i$ such that $y \not\sim_L z$. We know that $x_i \sim_L y$ and $x_i \sim_L z$ by the definition of X_i. We have the following three cases: (1) $|x_i| < |y|, |z|$, (2) $|y| \leq |x_i| \leq |z|$ (or $|z| \leq |x_i| \leq |y|$), and (3) $|x_i| > |y|, |z|$. If (1) or (2), then $y \sim_L z$ by Lemma 1. This would contradict our assumption. If (3), then it is easy to prove that $y \not\sim x_j$ and $z \not\sim x_j$, for all $j \neq i$, using Lemma 1 and the definition of X_i. Then we can replace x_i by both y and z to obtain a longer dissimilar sequence $[x_1, \ldots, x_{i-1}, y, z, x_{i+1}, \ldots, x_n]$. This contradicts the fact that $[x_1, \ldots, x_{i-1}, x_i, x_{i+1}, \ldots, x_n]$ is a maximal dissimilar sequence of L. Hence, $y \sim z$ and X_i is a similarity set. □

Corollary 1 *For each finite language L, there is a unique number $N(L)$ which is the number of elements in any canonical dissimilar sequence of L.*

Theorem 2 *Let S_1 and S_2 be two L-similarity sets and x_1 and x_2 the shortest words in S_1 and S_2, respectively. If $x_1 \sim_L x_2$ then $S_1 \cup S_2$ is a L-similarity set.*

Proof. In order to prove that $S_1 \cup S_2$ is a L-similarity set, it suffices to show that for an arbitrary word $y_1 \in S_1$ and an arbitrary word $y_2 \in S_2$, $y_1 \sim_L y_2$ holds. Without loss of generality, we assume that $|x_1| \leq |x_2|$. We know that $|x_1| \leq |y_1|$ and $|x_2| \leq |y_2|$. Since $x_1 \sim_L x_2$ and $x_2 \sim_L y_2$, we have $x_1 \sim_L y_2$ (Lemma 1 (2)), and since $x_1 \sim_L y_1$ and $x_1 \sim_L y_2$, we have $y_1 \sim_L y_2$ (Lemma 1 (1)). □

In the following, we define the similarity relation on the set of states of a DFA or a DFCA. Note that if a DFA A accepts a finite language L, then A is also a DFCA for L.

For a DFA (or a DFCA) A, we use the notation $A = (Q, \Sigma, \delta, s, F)$, where Q denotes the finite set of states, Σ the finite alphabet, $\delta : Q \times \Sigma \to Q$ the transition function, $s \in Q$ the initial state, and $F \subseteq Q$ the final state set.

Definition 5 *Let $A = (Q, \Sigma, \delta, 0, F)$ be a DFA (or a DFCA). We define, for each state $q \in Q$,*

$$level(q) = min\{|w| \mid \delta(0, w) = q\},$$

i.e., level(q) is the length of the shortest path (in the directed graph associated with the automaton) from the initial state to q.

Let $A = (Q, \Sigma, \delta, 0, F)$ be a DFA. For each state $q \in Q$, we denote $x_A(q) = \min\{w \mid \delta(0, w) = q\}$, where the minimum is taken according to the lexicographic ordering, and $L_A(q) = \{w \in \Sigma^* \mid \delta(q, w) \in F\}$. When the automaton A is understood, we write x_q instead of $x_A(q)$ and L_q instead $L_A(q)$. The length of x_q is equal to $level(q)$, therefore $level(q)$ is defined for each $q \in Q$.

Definition 6 *Let $A = (Q, \Sigma, \delta, 0, F)$ be a DFCA for a finite language L with l being the longest word(s) in L. Let $p, q \in Q$ and $m = \max\{level(p), level(q)\}$. We say that $p \sim_A q$ if for every $w \in \Sigma^{\leq l-m}$, $\delta(p, w) \in F$ iff $\delta(q, w) \in F$.*

We use the notation $p \sim q$ instead of $p \sim_A q$ whenever L is clearly understood from the context.

Definition 7 *Let $A = (Q, \Sigma, \delta, 0, F)$ be a DFA or a DFCA for a finite language L with l being the length of the longest word(s) in L. For $p \in Q$, denote by x_p a shortest word in Σ^* such that $\delta(0, x_p) = p$; x_p is called a "representative" of p.*

Using the previous definitions we can observe some simple properties:

Theorem 3 *Let $A = (Q, \Sigma, \delta, 0, F)$ be a DFCA of a finite language L and $p, q \in Q$. Then $p \sim_A q$ if and only if $x_p \sim_L x_q$.*

Proof. Let l denote the length of the longest word(s) in L. Let $level(p) = i$ and $level(q) = j$. Clearly, $|x_p| = i$ and $|x_q| = j$ by definition. Let $m = \max\{i, j\}$. Then the condition $w \in \Sigma^{\leq l-m}$ is the same as the condition $w \in \Sigma^*$ such that both $|x_p w| \leq l$ and $|x_q w| \leq l$. We know that, by definition, $p \sim_A q$ means that $\delta(p, w) \in F$ iff $\delta(q, w) \in F$ for each $w \in \Sigma^{\leq l-m}$; and $x_p \sim_L x_q$ means that $x_p w \in L$ iff $x_q w \in L$ for each $w \in \Sigma^*$ such that $|x_p w| \leq l$ and $|x_q w| \leq l$. Then, clearly, $p \sim_A q$ if and only if $x_p \sim_L x_q$. □

All words that reach the same state in a DFA or a DFCA are similar.

Lemma 2 *Let $A = (Q, \Sigma, \delta, 0, F)$ be a DFCA of a finite language L. Let $q \in Q$ and $x, y \in \Sigma^*$ such that $\delta(0, x) = \delta(0, y) = q$. Then $x \sim_L y$.*

Proof. Clearly, for all $w \in \Sigma^*$, including those w such that $|xw| \leq l$ and $|yw| \leq l$, $\delta(0, xw)$ and $\delta(0, yw)$ reach exactly the same state. So, $xw \in L$ iff $yw \in L$. □

Similarity on states implies similarity on words:

Lemma 3 *Let $A = (Q, \Sigma, \delta, 0, F)$ be a DFCA of a finite language L. Let $u, v \in \Sigma^*$ such that $\delta(0, u) = p$ and $\delta(0, v) = q$. If $p \sim_A q$ then $u \sim_L v$.*

Proof. Since $p \sim_A q$, we have $x_p \sim_L x_q$ by Theorem 3. By Lemma 2, we know that $x_p \sim_L u$ and $x_q \sim_L v$. Therefore, $x_q \sim_L u$ and then $u \sim_L v$ by using (1) of Lemma 1. □

Note that similarity on words does not necessarily imply similarity on states. The following is a counter example.

Example: Let $L = \{a, b, aa, aaa, bab\}$. Then $l = 3$. The following is a DFCA of L. Let $x = b$ and $y = bab$. Clearly, $x \sim_L y$. However, $\delta(0, x) \not\sim \delta(0, y)$.

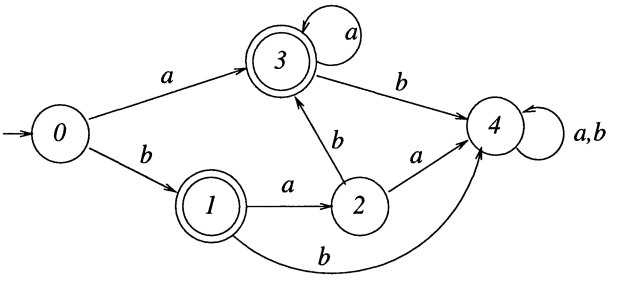

Fig. 3. If $x \sim_L y$ then it is not necessarily true that $\delta(0,x) \sim_A \delta(0,y)$.

It always holds that $x \sim_L y$ implies $\delta(0,x) \sim_A \delta(0,y)$ if both x and y are the shortest words that, from the initial state, reach their destination states, respectively.

Lemma 4 *Let $A = (Q, \Sigma, \delta, 0, F)$ be a DFCA of a finite language L. Let $s, p, q \in Q$ such that $level(s) = i$, $level(p) = j$, $level(q) = m$, $i \leq j \leq m$. The following statements are true:*

1. *If $s \sim_A p$, $s \sim_A q$, then $p \sim_A q$.*
2. *If $s \sim_A p$, $p \sim_A q$, then $s \sim_A q$.*
3. *If $s \sim_A p$, $p \not\sim_A q$, then $s \not\sim_A q$.*
4. *If $s \sim_A p$, $s \not\sim_A q$, then $p \not\sim_A q$.*

Proof. We apply Lemma 1 and Theorem 3. □

We are now ready to state the theorem that is the basis for minimizing DFCAs.

Theorem 4 *Let $A = (Q, \Sigma, \delta, 0, F)$ be a DFCA for a finite language L. Assume that $p \sim_L q$ for some $p, q \in Q$ such that $p \neq q$ and $level(p) \leq level(q)$. Then we can construct a DFCA A' for L in the following way: $A' = (Q', \Sigma, \delta', 0, F')$ where $Q' = Q - \{q\}$, $F' = F - \{q\}$, and*

$$\delta'(t,a) = \begin{cases} \delta(t,a) & \text{if } \delta(t,a) \neq q, \\ p & \delta(t,a) = q \end{cases}$$

for each $t \in Q'$ and $a \in \Sigma$.

Proof. It suffices to prove that A' is a DFCA of L. Let l be the length of the longest word(s) in L and assume that $level(p) = i$ and $level(q) = j$, $i \leq j$. Consider a word $w \in \Sigma^{\leq L}$. We now prove that $w \in L$ iff $\delta'(0,w) \in F'$.

If there is no prefix w_1 of w such that $\delta(0,w_1) = q$, then clearly $\delta'(0,w) \in F'$ iff $\delta(0,w) \in F$. Otherwise, let $w = w_1 w_2$ where w_1 is the shortest prefix of w such that $\delta(0,w_1) = q$. In the remaining, it suffices to prove that $\delta'(p,w_2) \in F'$ iff $\delta(q,w_2) \in F$. We prove this by induction on the length of w_2. First consider the case $|w_2| = 0$, i.e., $w_2 = \lambda$. Since $p \sim_A q$, $p \in F$ iff $q \in F$. Then $p \in F'$ iff $q \in F$ by the construction of A'. Thus, $\delta'(p,w_2) \in F'$ iff $\delta(q,w_2) \in F$. Suppose that the

statement holds for $|w_2| < l'$ for some $l' \leq l - |w_1|$. (Note that $l - |w_1| \leq l - j$.) Consider the case that $|w_2| = l'$. If there does not exist $u \in \Sigma^+$ such that $u \preceq_p w_2$ (u is a prefix of w_2) and $\delta(p,u) = q$, then $\delta(p,w_2) \in F - \{q\}$ iff $\delta(q,w_2) \in F - \{q\}$, i.e., $\delta'(p,w_2) \in F'$ iff $\delta(q,w_2) \in F$. Otherwise, let $w_2 = uv$ and u be the shortest nonempty prefix of w_2 such that $\delta(p,u) = q$. Then $|v| < l'$ (and $\delta'(p,u) = p$). By induction hypothesis, $\delta'(p,v) \in F'$ iff $\delta(q,v) \in F$. Therefore, $\delta'(p,uv) \in F'$ iff $\delta(q,uv) \in F$. □

Definition 8 *A DFCA $A = (Q,\Sigma,\delta,0,F)$ for a finite language L is said to be a minimal DFCA for L if, for any DFCA $A = (Q',\Sigma,\delta',0',F')$ for L, $\#Q \leq \#Q'$*

For a finite language, there may be several minimal DFCA that have different structures. The minimal Deterministic Finite Cover Automata for a finite language are not unique up to an isomorphism (i.e., renaming of the states).

Using the previous theorem we can easily prove the following corollary.

Corollary 2 *A DFCA A for a finite language L is a minimal DFCA for L if and only if no two distinct states of A are similar.*

Proof. Let $A = (Q,\Sigma,\delta,0,F)$ be a DFCA for L. If there exist two distinct states $p,q \in Q$ such that $p \sim q$, then A is not minimal by Theorem 4. For the other direction of the proof, if A is not minimal, then there exists a DFCA $A' = (Q',\Sigma,\delta',0',F')$ of L such that $\#Q' < \#Q$. Then there exist $p,q \in Q$ such that x_p and x_q reach the same state in A'. By Lemma 2, we have $x_p \sim_L x_q$. Therefore, $p \sim_A q$ by Theorem 3. □

Lemma 5 *Let A be a DFCA of L and $L' = L(A)$. Then $x \equiv_{L'} y$ implies $x \sim_L y$.*

Proof. Let l be the length of the longest word(s) in L. Let $x \equiv_{L'} y$. So, for each $z \in \Sigma^*$, $xz \in L'$ iff $yz \in L'$. We now consider all words $z \in \Sigma^*$, such that $|\, xz \,| \leq l$ and $|\, yz \,| \leq l$. Since $L = L' \cap \Sigma^{\leq l}$ and $xz \in L'$ iff $yz \in L'$, we have $xz \in L$ iff $yz \in L$. Therefore, $x \sim_L y$ by the definition of $\sim_L$. □

Corollary 3 *Let $A = (Q,\Sigma,\delta,0,F)$ be a DFCA of a finite language L and $L' = L(A)$ (L' is the language accepted by A as a DFA not as a DFCA). Then $p \equiv_A q$ (A as a DFA) implies $p \sim_A q$ (A as a DFCA).*

Corollary 4 *A minimal DFCA of L is a minimal DFA.*

Proof. Let $A = (Q,\Sigma,\delta,0,F)$ be a minimal DFCA of a finite language L. Suppose that A is not minimal as a DFA for $L(A)$, then there exists $p,q \in Q$ such that $p \equiv_{L'} q$, then $p \sim_A q$. By Theorem 4 it follows that A is not a minimal DFCA, contradiction. □

Theorem 5 *Any minimal DFCA of L has exactly $N(L)$ states.*

Proof. Let $A = (Q,\Sigma,\delta,0,F)$ be a minimal DFCA of a finite language L and $\#Q = n$. Suppose that $n > N(L)$. Then there exist $p,q \in Q$, $p \neq q$, such that $x_p \sim_L x_q$ (because of the definition of $N(L)$). Then $p \sim_A q$ by Theorem 3. Thus, A is not minimal by Theorem 4. A contradiction.

Suppose that $N(L) > n$. Let $[y_1,\ldots,y_{N(L)}]$ be a canonical dissimilar sequence of L. Then there exist i,j, $1 \leq i,j \leq N(L)$ and $i \neq j$, such that $\delta(0,y_i) = \delta(0,y_j) = q$ for some $q \in Q$. Then $y_i \sim_L y_j$. Again a contradiction.

Therefore, we have $n = N(L)$. □

3. The New Algorithm

Given a DFA that accepts a finite language, we can construct a minimal DFCA for the given language in two steps: (1) compute the similarity relation between the states of the DFA, and (2) merge similar states. Note that the similarity relation is not transitive. So, if $p \sim q$ and $q \sim r$, we cannot simply merge p, q, and r together in general. Step (1) is the most complex one. A naive algorithm for determine whether $p \sim q$ is to check whether $\delta(p,z), \delta(q,z) \in F$ or $\delta(p,z), \delta(q,z) \in Q - F$ for all words z such that $|z| \leq l - \max(level(p), level(q))$. This would need exponential time. In the algorithm given in [3], it needs $O(n^2)$ time to determine whether two states are similar. The time complexity of the entire step (1) of that algorithm is $O(n^4)$.

Here we use a different approach and the time complexity of our algorithm is $O(n^2)$. In this section, we will describe our new algorithm. However, before describing the algorithm, we have to give several new definitions and prove several new results.

3.1. Some More Definitions and Results

Again we assume that $A = (Q, \Sigma, \delta, 0, F)$ is a DFA accepting a finite language L over the alphabet Σ and l is the length of the longest word(s) in L. We assume that A is a complete DFA and there is no useless state in Q except the sink state d, i.e., for each $q \in Q - \{d\}$, there exist $u, v \in \Sigma^*$ such that $\delta(0,u) = q$ and $\delta(q,v) \in F$.

Definition 9 *For $p, q \in Q$ and $p \neq q$, we define*

$$range(p,q) = l - \max\{level(p), level(q)\}.$$

Intuitively, $range(p,q)$ is the maximum length of a word w that satisfies both $|x_p w| \leq l$ and $|x_q w| \leq l$.

Definition 10 *Let $p, q \in Q$ and $z \in \Sigma^*$. We say that p and q fail on z if $\delta(p,z) \in F$ and $\delta(q,z) \in Q - F$ or vice versa, and $|z| \leq range(p,q)$.*

Theorem 6 *$p \not\sim q$ if and only if there exists $z \in \Sigma^*$ such that p and q fail on z.*

Definition 11 *If $p \not\sim q$, we define*

$$gap(p,q) = min\{|z| \mid p \text{ and } q \text{ fail on } z\}.$$

If $p \not\sim q$, then $gap(p,q)$, intuitively, is the length of the shortest word(s) that can show that p and q are dissimilar. It is clear that $gap(p,q) = gap(q,p)$ and $gap(p,q) < l$ for any $p, q \in Q$ such that $p \not\sim q$. For convenience, we define $gap(p,q) = l$ if $p \sim q$. The next theorem is clear.

Theorem 7

(1) Let d be the sink state of A. If $level(d) > l$, then $d \sim q$ for each $q \in Q - \{d\}$. If $level(d) \leq l$, then $d \not\sim f$ and $gap(d,f) = 0$ for each $f \in F$.

(2) If $p \in F$ and $q \in Q - F - \{d\}$ or vice versa, then $p \not\sim q$ and $gap(p,q) = 0$.

Lemma 6 *Let $p, q \in Q$, $p \neq q$, and $r = \delta(p, a)$ and $t = \delta(q, a)$, for some $a \in \Sigma$. Then $range(p, q) \leq range(r, t) + 1$.*

Proof. It is clear that $level(r) \leq level(p) + 1$ and $level(t) \leq level(q) + 1$. So,

$$\max(level(r), level(t)) \leq \max(level(p), level(q)) + 1.$$

Then, by Definition 9, $range(p, q) \leq range(r, t) + 1$. □

Theorem 8 *Let p and q be two states such that either $p, q \in F$ or $p, q \in Q - F$. Then $p \not\sim q$ if and only if there exists $a \in \Sigma$ such that $\delta(p, a) = r$ and $\delta(q, a) = t$, $r \not\sim t$, and*

$$gap(r, t) + 1 \leq range(p, q).$$

Proof. *Only if*: We assume that $p \not\sim q$ and will show that there exists a pair (r, t) satisfying the conditions of the theorem. Choose $z \in \Sigma^*$ such that p and q fail on z and $|z| = gap(p, q)$. Note that $|z| > 0$ because of the given condition of p and q. By the definition of *gap* function, we know that $|z| \leq range(p, q)$. Without loss of generality, we assume that $\delta(p, z) \in F$ and $\delta(q, z) \in Q - F$. Let $z = az'$. Then $\delta(p, az') = \delta(r, z') \in F$ and $\delta(q, az') = \delta(t, z') \in Q - F$ for some $r, t \in Q$. By Lemma 6, we know that $range(r, t) \geq range(p, q) - 1$. Then $|z'| \leq range(r, t)$. By Definition 10, r and t fail on z' and $r \not\sim t$. Since $gap(r, t) \leq |z'|$, we have $gap(r, t) + 1 \leq range(p, q)$.

If: Assume that there exists $a \in \Sigma$ such that $\delta(p, a) = r$, $\delta(q, a) = t$, $r \not\sim t$, and $gap(r, t) + 1 \leq range(p, q)$. Then there is $z' \in \Sigma^*$ such that r and t fail on z' and $|z'| = gap(r, t)$. Let $z = az'$. Then $|z| = gap(r, t) + 1$ and thus $|z| \leq range(p, q)$. Therefore, p and q fail on z. In other words, $p \not\sim q$. □

The following theorem gives a formula which computes $gap(p, q)$ for two state p and q that are either both final states or both non-final states.

Theorem 9 *If $p \not\sim q$ such that $p, q \in F$ or $p, q \in Q - F$, then*

$$gap(p, q) = \min\{gap(r, t) + 1 \mid \delta(p, a) = r \text{ and } \delta(q, a) = t, \text{ for } a \in \Sigma, \\ r \not\sim t, \text{ and } gap(r, t) + 1 \leq range(p, q)\}.$$

Proof. We first prove that $gap(p, q) \leq gap(r, t) + 1$ for every pair $r, t \in Q$ such that $\delta(p, a) = r$, $\delta(q, a) = t$, $r \not\sim t$, and $gap(r, t) + 1 \leq range(p, q)$. Let (r, t) be an arbitrary pair that satisfy the above conditions. Since $r \not\sim t$, there exists z' such that r and t fail on z' and $|z'| = gap(r, t)$. It is also clear that $|az'| \leq range(p, q)$ since $gap(r, t) + 1 \leq range(p, q)$. Then p and q fail on $z = az'$. By definition, $gap(p, q) \leq |z|$. So, we have $gap(p, q) \leq gap(r, t) + 1$.

We now prove the other direction, i.e., there exist $r, t \in Q$ such that $\delta(p, a) = r$, $\delta(q, a) = t$, $r \not\sim t$, and $gap(r, t) + 1 \leq gap(p, q)$. Let $z \in \Sigma^*$ such that p and q fail on z and $|z| = gap(p, q)$. Clearly, $|z| > 0$ by the given conditions. Then $z = az'$ for some $a \in \Sigma$. Let $\delta(p, a) = r$ and $\delta(q, a) = t$. Then $range(p, q) \leq range(r, t) + 1$ by Lemma 6. Thus, $|z'| \leq range(r, t)$. Then clearly r and t fail on z'. By definition, $gap(r, t) \leq |z'|$. Then we have $gap(r, t) \leq |z| - 1 = gap(p, q) - 1$. □

3.2. The Algorithm

The algorithm consists of two main parts: the first is to determine the similarity relation between states; the second is to merge similar states.

In the first part of the algorithm, we determine the similarity relation by computing the *gap* function starting from the sink state and along the inverse direction of the transitions of the given DFA. Note that the construction of a minimal DFCA is different from the minimization of an acyclic DFA. In the latter case, if two states have different *heights* then they are not equivalent. (The height of a state is the length of the longest path starting from this state to a final state.) However, in the former, it is possible that two states are similar even if they have different heights. The equivalence relation is a refinement of the similarity relation with respect to a finite language.

In the following, we assume that the given DFA accepting a finite language is complete (a transition is defined for each state and each letter in the alphabet) and reduced (no useless states except one sink state). We also assume that the given DFA is ordered, i.e., the $n+1$ states (including the sink state) of the DFA are numbered by $0, 1, \ldots, n$ such that there is no transition from state j to state i if $0 \leq i < j \leq n$. This implies that 0 is the starting state, n is the sink state, and *n-1* is the last final state. All the above pre-conditions can be achieved in linear time in terms of the number of states of the given DFA. Note that the size of a DFA is linear to its number of states.

Algorithm for computing the *gap* function

Input: An ordered, reduced, and complete DFA $A = (Q, \Sigma, \delta, 0, F)$, with $n+1$ states, which accepts a finite language L, and the length l of the longest word in L

Output: $gap(i, j)$ for each pair $i, j \in Q$ and $i < j$

Algorithm:

1. For each $i \in Q$ compute $level(i)$;

2. //initialize $gap(i, n)$ for each $i \in Q - \{n\}$

   for $i = 0$ to $n - 1$ **do** $gap(i, n) = l$ **end for**;
   if $level(n) \leq l$ **then**
   for each $i \in F$ **do** $gap(i, n) = 0$ **end for**
   end if;

3. //initialize $gap(i, j)$ for $i, j \in Q - \{n\}$ and $i < j$

   for each pair $i, j \in Q - \{n\}$ such that $i < j$ **do**
   if $i \in F$ and $j \in Q - F$ or vice versa **then**
   $gap(i, j) = 0$;
   else
   $gap(i, j) = l$;
   end if;
   end for;

4. //compute $gap(i,j)$ for $0 \leq i \leq n-2$, $i < j \leq n$

```
for i = n − 2 down to 0 do
    for j = n down to i + 1 do
        for each a ∈ Σ do
            let i' = δ(i, a) and j' = δ(j, a);
            if i' ≠ j' then
                g = if (i' < j') then gap(i', j') else gap(j', i');
                if g + 1 ≤ range(i, j) then
                    gap(i,j) = min(gap(i,j), g+1);
                end if;
            end if;
        end for;
    end for;
end for
```

Algorithm for merging all the similar states

Input: An ordered, reduced, and complete DFA $A = (Q, \Sigma, \delta, 0, F)$ which accepts a finite language L, l the length of the longest word in L, and $gap(i,j)$ for each pair $i, j \in Q$ and $i < j$

Output: A minimal DFCA A' for L

Algorithm:

1. Let $P[0..n]$ be a Boolean array with each $P[i]$, $0 \leq i \leq n$, initialized to $false$;

2.
```
for i = 0 to n − 1 do
    if P[i] == false then
        for j = i + 1 to n do
            if P[j] == false and gap(i, j) = l then
                merge j to i;
                P[j] = true;
            end if;
        end for;
    end if;
end for.
```

3. Eliminate the unreachable states in the new automaton.

For convenience, we assume that the number of states is $n+1$ in the above algorithm and there is at least one state in A. Thus $n = 0$ if there is only one state in A. In the above algorithm, by "merge j to i" we mean that all the transitions with j as the destination are changed to transitions with i as the destination and we delete j, i.e., following the steps described in Theorem 4.

We now consider the correctness of the two algorithms. First we consider the algorithm that computes the *gap* function. One can see that the idea of the algorithm is to initialize the *gap* function between a final state and a nonfinal state (which is 0), and then compute the *gap* function of all pair of states backwards (from states of higher number to that of lower number). The computation is correct, which can be

seen in two steps. The initiation is correct because of Theorem 7, and the remaining is given by Theorem 9 and Theorem 8.

Let us now consider the second algorithm: merging the similar states to obtain a minimal DFCA.

One can see that after merging two states it is possible that some part of the automaton becomes unreachable (since the transitions to some states are re-directed towards the other state) and also some states can have their level increased. We may just erase the unreachable part and continue our algorithm on the reachable part of the automaton. However, the checking of reachability takes time and it makes the algorithm more complex. We show that the merging of all similar states that relation was computed in the first algorithm will produce a minimal DFCA.

Note that after each merging, we have the following possible cases for the next merging: Case 1: we merge a reachable state into another reachable state. Case 2: merge a reachable state into an unreachable state. Case 3: merge an unreachable state into a reachable one. Case 4: merge two unreachable states. In whatever cases, we can see that for states i and j, if $i \sim j$ originally and they are not merged, then it is still true that for every $w \in \Sigma^{\leq l-m}$, $\delta(i,w) \in F$ iff $\delta(j,w) \in F$, where $m = \max(level(i), level(j))$ and the levels are original levels.

This means that we can iterate the merging of the states. So the algorithm is correct.

Clearly, both algorithms have the time complexity $O(n^2)$. Then we have the following.

Theorem 10 *The above two algorithms are correct and both have the time complexity* $O(n^2)$.

Example Let $A = (Q, \Sigma, \delta, s, F)$ be a DFA shown in Figure 4. Clearly, A accepts the finite language $L = \{abc, ababc, abababc\}$, where $l = 7$. A is the same DFA as the one shown in Figure 1 except that the sink state is explicitly shown here. We

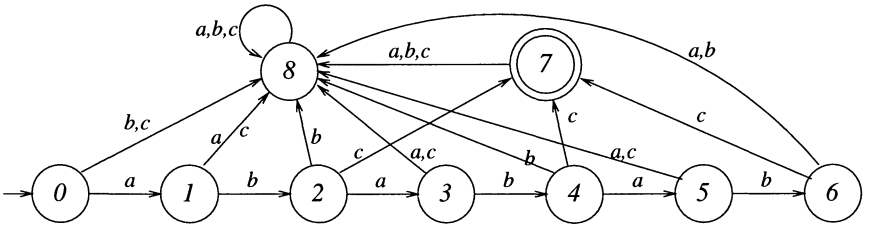

Fig. 4. Example: A DFA for a finite language L.

follow the first algorithm to calculate the *gap* function. At Step (1), we calculate $level(i)$ for each $i \in Q$:

state	0	1	2	3	4	5	6	7	8
level	0	1	2	3	4	5	6	3	1

After (2), (3), and (4), we have $gap(i,j)$ for each $0 \leq i \leq 7$, $1 \leq j \leq 8$, and $i < j$ as follows:

	1	2	3	4	5	6	7	8
0	2	1	2	1	2	1	0	3
1		1	7	1	7	1	0	2
2			1	7	1	7	0	1
3				1	7	1	0	2
4					1	7	0	1
5						1	0	2
6							0	1
7								0

In the above table, states i and j, $i < j$, are similar if $gap(i, j) = 7$. Then we follow the second algorithm to merge all the similar states and obtain a minimal DFCA for L, which is shown in Figure 5.

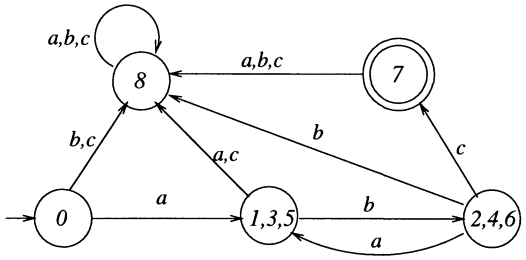

Fig. 5. A minimal DFCA for L.

4. Concluding Remarks

We have shown an $O(n^2)$ algorithm for constructing a minimal DFCA for a finite language given in the form of a DFA. This is a significant improvement from the $O(n^4)$ algorithm given in [3]. This new algorithm is also much easy to comprehend and implement. The algorithm can be modified into a minimization algorithm for general DFCA.

In the future, we will conduct more experiments on DFCA with finite languages from real-world applications. It is important to know how much reduction on the size of the automata one can achieve by using DFCA instead of DFA. We believe that the reduction can be large for certain types of applications, but minor on others.

Acknowledgements

This research is supported by the Natural Sciences and Engineering Research Council of Canada grant OGP0041630 and a graduate scholarship.

References

1. C. Campeanu, K. Culik, K. Salomaa, S. Yu, "State Complexity of Basic Operations on Finite Languages", *Proceedings of the Fourth International Workshop on Implementing Automata* VIII 1-11, 1999, to appear in Springer LNCS.

2. C. Campeanu, K. Salomaa, S. Yu, "State Complexity of Regular Languages: Finite Versus Infinite", *Finite vs Infinite – Contributions to an Eternal Dilemma*, edited by C. Calude and G. Paun, Springer 2000, pps. 53-73.
3. C. Campeanu, N. Santean, S. Yu, "Minimal Cover-Automata for Finite Languages", *Proceedings of the Third International Workshop on Implementing Automata* (WIA'98) 1998, 32-42. An extended version entitled "Finite Languages and Cover-Automata" is accepted by *Theoretical Computer Science.*
4. J.-M. Champarnaud and D. Maurel, *Automata Implementation*, Third International Workshop on Implementing Automata, LNCS 1660, Springer, 1999.
5. C. Dwork and L. Stockmeyer, "A Time Complexity Gap for Two-Way Probabilistic Finite-State Automata", *SIAM Journal on Computing*, vol.19 (1990) 1011-1023.
6. M.R. Garey and D.S. Johnson, *Computer and Intractability — A Guide to the Theory of NP-Completeness*, Freeman, 1979.
7. J. Kaneps, R. Frievalds, "Running Time to Recognize Non-Regular Languages by 2-Way Probabilistic Automata", *in ICALP'91, LNCS*, Springer-Verlag, New-York/Berlin (1991) vol 510, 174-185.
8. A. Salomaa and S. Yu, "Decomposition of Finite Languages", *Proceedings of Fourth International Conference on Developments in Language Theory* (1999) 8-20.
9. N. Santean, *Towards a Minimal Representation for Finite Languages: Theory and Practice*, MSc Thesis, Department of Computer Science, The University of Western Ontario, 2000.
10. B.A. Trakhtenbrot and Ya.M. Barzdin, *Finite Automata — Behavior and Synthesis*, North-Holland/American Elsevier, 1973.
11. D. Wood and S. Yu, *Automata Implementation*, Second International Workshop on Implementing Automata, LNCS 1436, Springer, 1998.