# An $O(n^2)$ Algorithm for Constructing Minimal Cover Automata for Finite Languages*

Andrei Păun, Nicolae Sântean, and Sheng Yu

Department of Computer Science, University of Western Ontario
London, Ontario, Canada N6A 5B7
{apaun, santean, syu}@csd.uwo.ca

**Abstract.** Cover automata were introduced in [1] as an efficient representation of finite languages. In [1], 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 introduce a new efficient transformation algorithm with the time complexity $O(n^2)$, which is a significant improvement from the previous algorithm.

## 1 Introduction

Finite languages have many practical applications [6,2]. However, the finite languages used in applications are generally very large, which need thousands or even millions of states if represented by deterministic finite automata (DFA) or similar structures. In [1], 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 [5].

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 $\Sigma$. 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 complete). A minimal DFCA is shown in Figure 2, which has only 4 states (5 if complete).

In [1], an algorithm was given for constructing a minimal DFCA from a given DFA that accepts a finite language. The time complexity of the algorithm

* This work has been partially supported by the Natural Sciences and Engineering Research Council of Canada grants OGP0041630 and a graduate scholarship.

S. Yu and A. Păun (Eds.): CIAA 2000, LNCS 2088, pp. 243–251, 2001.

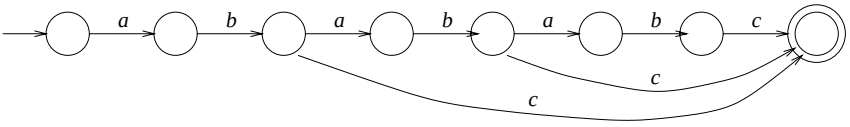


**Fig. 1.** The minimal DFA accepting $L$

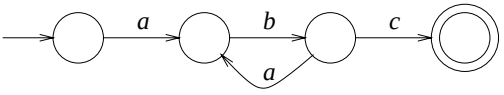


**Fig. 2.** A minimal DFCA for $L$ with $l = 7$

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. 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 [1] in time complexity, but also much easier to comprehend and to implement.

The two algorithms differ 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 [1]. A new algorithm is also given for merging similar states, which is simpler than the one given in [1]. We also prove several new theorems on the similarity relation 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 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 [1] or [5].

Let $S$ be a finite set and $n$ a nonnegative integer. By $S^{\leq n}$ we denote $\cup_{i=0}^{n} S^i$.

**Definition 1.** *Let $L \subset \Sigma^*$ be a finite language over an alphabet $\Sigma$ and $l$ the length of the longest word(s) in $L$. A language $L'$ over $\Sigma$ 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. We first define the similarity relation on words 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 [4], and then studied in [3], [1], [5], 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 $\Sigma$ and $l$ the length of the longest word(s) in $L$. Let $x, y \in \Sigma^*$. We define the following relation:*

*(1) $x \sim_L y$ if for all $z \in \Sigma^*$ such that $\mid xz \mid \leq l$ and $\mid yz \mid \leq l, xz \in L$ iff $yz \in L$;*
*(2) $x \not\sim_L y$ 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 $x \not\sim_L z$.*

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

1. *A sequence of words $(x_1, \ldots, x_n)$ over $\Sigma$ 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 $i \neq j$.*
2. *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 $L$, $m \leq n$.*

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 $L$.

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

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

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

**Definition 6.** *Let $A = (Q, \Sigma, \delta, s, F)$ 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.

We are now ready to state the theorem that is the basis for any algorithm for the minimization of DFCAs.

**Theorem 1.** *Let $A = (Q, \Sigma, \delta, s, F)$ 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 $A' = (Q', \Sigma, \delta', s, F')$ for $L$ such that $Q' = Q - \{q\}$, $F' = F - \{q\}$, and*

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

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

**Definition 7.** *A DFCA $A$ for a finite language is a minimal DFCA if and only if no two different states of $A$ are similar.*

**Theorem 2.** *For a finite language $L$, there is a unique number $N(L)$ such that any minimal DFCA for $L$ has exactly $N(L)$ states.*

Please refer to [1] for the proofs of the above theorems.

**Definition 8.** *Let $A = (Q, \Sigma, \delta, s, F)$ be a DFA or a DFCA for a finite language $L$ with $l$ be the length of the longest word(s) in $L$. For $p \in Q$, denote by $x_p$ a shortest word in $\Sigma^*$ such that $\delta(s, x_p) = p$; $x_p$ is called a "representative" of $p$.*

Note that for each $q \in Q$, $|x_q| = level(q)$.

**Theorem 3.** *$p \sim q$ if and only if $x_p \sim x_q$.*

One may refer to [1] for a proof.

## 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 [1], 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 New Definitions and Results

Again we assume that $A = (Q, \Sigma, \delta, s, F)$ is a DFA accepting a finite language $L$ over the alphabet $\Sigma$ 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 $\delta(s, u) = q$ 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 4.** *$p \not\sim q$ 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 5.**

*(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 2.** *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 6.** *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 2, 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 7.** *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 2. 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 a 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 $level(i)$ end for;
2. for $i = 0$ to $n - 1$ do $gap(i, n) = l$ end for;
   if $level(n) \leq l$ then
   for each $i \in F$ $gap(i, n) = 0$ end for
   end if;
3. for each pair $i, j \in Q - \{n\}$ such that $i < j$
   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. for $i = n - 2$ down to 0 do
   for $j = n$ down to $i + 1$ do
   for each $a \in \Sigma$ do
   let $i' = \delta(i, a)$ and $j' = \delta(j, a)$;
   if $i' \neq j'$ 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 similar states**
**Input**: A ordered, reduced, and complete DFA $A = (Q, \Sigma, \delta, 0, F)$ which accepts a finite language $L$, 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.

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 $A$.

The step "merge $j$ to $i$;" follows the steps described in Theorem 1.

The correctness of the algorithm can be easily established with Theorem 5, Theorem 6, and Theorem 7. So, we omit the formal proof here.

We now consider the time complexity of the algorithm. In the first part, each of Step 1 and Step 2 is $O(n)$. Clearly, Step 3 takes $O(n^2)$ iterations. Step 4 is the main part, which has two nested loops, each of which has $O(n)$ iterations. Each inner iteration is $O(|\Sigma|)$, where $|\Sigma|$ is a constant. Therefore, the first part of the algorithm, that computes the *gap* function, is $O(n^2)$. Clearly, the second part is also $O(n^2)$. So, the time complexity of the algorithm is $O(n^2)$.

## 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 [1]. 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.

## References

1. 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.
2. J.-M. Champarnaud and D. Maurel, *Automata Implementation*, Third International Workshop on Implementing Automata, LNCS 1660, Springer, 1999.
3. 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.
4. 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.
5. N. Santean, *Towards a Minimal Representation for Finite Languages: Theory and Practice*, MSc Thesis, Department of Computer Science, The University of Western Ontario, 2000.
6. D. Wood and S. Yu, *Automata Implementation*, Second International Workshop on Implementing Automata, LNCS 1436, Springer, 1998.