

RESEARCH ARTICLE

Efficient Deterministic Finite Automata Minimization Based on Backward Depth Information

Desheng Liu*, Zhiping Huang, Yimeng Zhang, Xiaojun Guo, Shaojing Su

College of Mechatronics and Automation, National University of Defense Technology, ChangSha 410073, China

* deshengcyzw@163.com

Abstract

Obtaining a minimal automaton is a fundamental issue in the theory and practical implementation of deterministic finite automata (DFAs). A minimization algorithm is presented in this paper that consists of two main phases. In the first phase, the backward depth information is built, and the state set of the DFA is partitioned into many blocks. In the second phase, the state set is refined using a hash table. The minimization algorithm has a lower time complexity $O(n)$ than a naive comparison of transitions $O(n^2)$. Few states need to be refined by the hash table, because most states have been partitioned by the backward depth information in the coarse partition. This method achieves greater generality than previous methods because building the backward depth information is independent of the topological complexity of the DFA. The proposed algorithm can be applied not only to the minimization of acyclic automata or simple cyclic automata, but also to automata with high topological complexity. Overall, the proposal has three advantages: lower time complexity, greater generality, and scalability. A comparison to Hopcroft's algorithm demonstrates experimentally that the algorithm runs faster than traditional algorithms.



OPEN ACCESS

Citation: Liu D, Huang Z, Zhang Y, Guo X, Su S (2016) Efficient Deterministic Finite Automata Minimization Based on Backward Depth Information. PLoS ONE 11(11): e0165864. doi:10.1371/journal.pone.0165864

Editor: Gui-Quan Sun, Shanxi University, CHINA

Received: May 25, 2016

Accepted: October 19, 2016

Published: November 2, 2016

Copyright: © 2016 Liu et al. This is an open access article distributed under the terms of the [Creative Commons Attribution License](https://creativecommons.org/licenses/by/4.0/), which permits unrestricted use, distribution, and reproduction in any medium, provided the original author and source are credited.

Data Availability Statement: All relevant data are within the paper and its Supporting Information file.

Funding: The author(s) received no specific funding for this work.

Competing Interests: The authors have declared that no competing interests exist.

Introduction

Finite automata, regular grammar, and regular expressions are three dissimilar representations for regular languages. Regular grammar and regular expressions generate regular languages, and finite automata is a computation model of speech recognition for regular languages [1]. Finite automata is widely used in areas such as text processing [2], compilation [3], pattern matching [4, 5], network intrusion detection and protection [6, 7], image analysis and spatial dynamics [8, 9]. Finite automata mainly have two alternative implementations: nondeterministic finite automata (NFAs) and deterministic finite automata (DFAs). DFAs are used widely because they have predictable and acceptable memory bandwidth requirements. Regular expressions are usually used to describe engineering requirements in practice, and they can be converted into a DFA by a sequence of operations [10, 11]. To save memory space, the minimization of a DFA is indispensable in practical applications. In this paper, we focus on the

minimization problem without considering the conversion from regular expressions or regular grammar to DFA.

It has been proven theoretically that a DFA has a unique minimal formalization, but only up to isomorphism [3]. This means that the minimal DFA is unique, and has the least number of states needed to recognize a language represented by regular expressions or regular grammar. The minimization of a DFA is beneficial not only for practical applications, but also mathematical problems in theory. Many algorithms can be used to minimize a DFA, and these can be classified into four categories.

The first category uses a series of refinements of a partition on the state set. A typical method is mentioned in [12], and has time complexity $O(n^2)$. Hopcroft's and Moore's algorithms also fall into this category. At present, Hopcroft's algorithm is the most efficient known algorithm and has a worst-case time of $O(n \log n)$ [13]. Several authors [14, 15] have proven the tightness of the upper bound of this complexity for different families of automata. Hopcroft's algorithm has recently been extended to handle incomplete DFAs [16, 17], and has now been re-described many times to make it easier to understand [18, 19]. Moore's algorithm is much simpler than Hopcroft's algorithm. However, it leads to a quadratic worst-case time complexity [20, 21].

The second type of minimization algorithm uses a sequence of merging states. The most famous algorithm of this type is Revuz's algorithm, which has a linear time complexity. The disadvantage of Revuz's algorithm is that it is constrained to acyclic automata [22]. Almeida and Zeitoun extended it to automata whose nontrivial strongly connected components are cycles [23, 24].

In the third category, Brzozowski's algorithm reverses and trims the automaton twice to obtain the minimal DFA [25]. Recently, studies on automaton minimization algorithms have mainly focused on incremental construction and dynamic DFA minimization, which is the fourth classification. Daciuk proposed an incremental algorithm for minimizing finite state automata that allows the minimization to be merged with the construction in a single step by adding new strings one by one and minimizing the resulting automaton on-the-fly [26]. Another incremental determinization algorithm for general finite automata called ISC is presented in [27]. Finally, the dynamic minimization solution proposed by Carrasco and Forcada keeps an automaton minimal when insertions or deletions are performed [28].

The previously proposed approaches have two main drawbacks. On the one hand, previous methods cannot be used on complicated automata; they are only suitable for acyclic automata, local automata, or other special automata. Automata that have distinguishable accepted states are generally used in many applications, such as network intrusion detection and image analysis. However, only Hopcroft's algorithm can minimize such DFAs. On the other hand, many algorithms are too complicated, slow, and difficult to implement. Along with the increases in size and topological complexity of DFAs in practice, the time consumed by minimization becomes increasingly important. In this paper, an efficient minimization algorithm is presented that can be used on various automata and is efficient in practice. The main contributions of this paper are as follows.

1. A minimization algorithm based on backward depth information is proposed. We define backward depth and prove the correctness of a partition based on this information.
2. Refinements based on a hash algorithm are introduced to obtain the final minimal DFA. The efficiency of this algorithm is explained and proven experimentally.
3. The proposed algorithm can obtain greater generality and time efficiency. Specifically, it can be efficiently implemented in scenarios where the automata has a more complicated topology complexity.

The remainder of this paper is organized as follows: firstly, some definitions and propositions related to the minimal DFA and proposed algorithm are stated. Then, we present the detailed steps for obtaining a minimal DFA: building backward depth information, hash table refinement with hash collision checking. In addition, the experiments and results for evaluating the efficiency of the algorithm are presented. Finally, we conclude the paper with a discussion on practical time efficiency, application range, and possible extensions.

Preliminaries

In order to illustrate the proposed minimization algorithm more clearly, some preliminaries that relate to the minimization of DFA are presented. First, some related concepts are defined, such as DFA, minimal DFA, and partition [12, 29]. Next, a series of propositions that are used in the algorithm are introduced, and the proofs can be found in [S1 File](#).

DFA. A DFA can be defined as a five-tuple $D = \langle Q, \Sigma, \delta, q_0, F \rangle$, where

- Q is a finite set of DFA states.
- Σ is a finite set of input symbols.
- $\delta: Q \times \Sigma \rightarrow Q$ is a transition function.
- $q_0 \in Q$ is the initial state.
- $F \subset Q$ is the accepted state set.

Given word $w = a_1 a_2 \cdots a_n (a_1, a_2, \cdots, a_n \in \Sigma)$ as the input, the current state will be transformed into the sequence according to transition function δ . This transition function can be extended into a function $\hat{\delta}: Q \times \Sigma^* \rightarrow Q$, where $\hat{\delta}(p, aw) = \hat{\delta}(\delta(p, a), w)$, Σ^* represents the set of words that composed by input symbols. If $\hat{\delta}(q_0, w) \in F$, then word w is recognized. A DFA can be expressed in other formalisms including a state transition table or state transition digraph. The state transition digraph and state transition table for regular expressions $RE = (ab.*cd, ef.*gh)$ are given in [Fig 1](#) and [Table 1](#), respectively. The dot-star notation “.” in the regular expressions represents any number of repetitions of any character. Because there are too many edges in the graph for the automata, unlabeled dashed edges represent the transitions that have the same next state for different symbols that have not been shown. In [Table 1](#), where S represents the states in the automata, the accepted states are bold, Σ denotes the symbol set, and the transitions are listed.

The function of DFA D is to recognize regular language L , and this can be formally expressed as $L(D) = \{w \in \Sigma^* | \hat{\delta}(q_0, w) \in F\}$. However, there are many DFAs that have different numbers of states for a regular expression set. If two DFAs D and D' recognize the same regular expression set, then D and D' are equivalent, which is denoted as $L(D) \equiv L(D')$. In the sense of isomorphism, there exists a minimal DFA that has the fewest number of states. A minimal DFA is defined as follows.

Minimal DFA. If DFA M is minimal, then there is no other DFA M' that has fewer states and is equivalent to DFA M .

The minimal DFA for any regular language is unique up to isomorphism, and this has been proved [3]. The transformation from a DFA to an equivalent minimal DFA is called the minimization. In a normal DFA, the essential cause of redundant states is that it has equivalent states. We present the definition of equivalent states as follows.

Equivalent States. Suppose that $p \in Q, q \in Q$, and $w \in \Sigma^*$ in DFA $D = \langle Q, \Sigma, \delta, q_0, F \rangle$ maintain $\hat{\delta}(p, w) = s_i$ and $\hat{\delta}(q, w) = s_j$. If states s_i and s_j are either both accepted states or non-

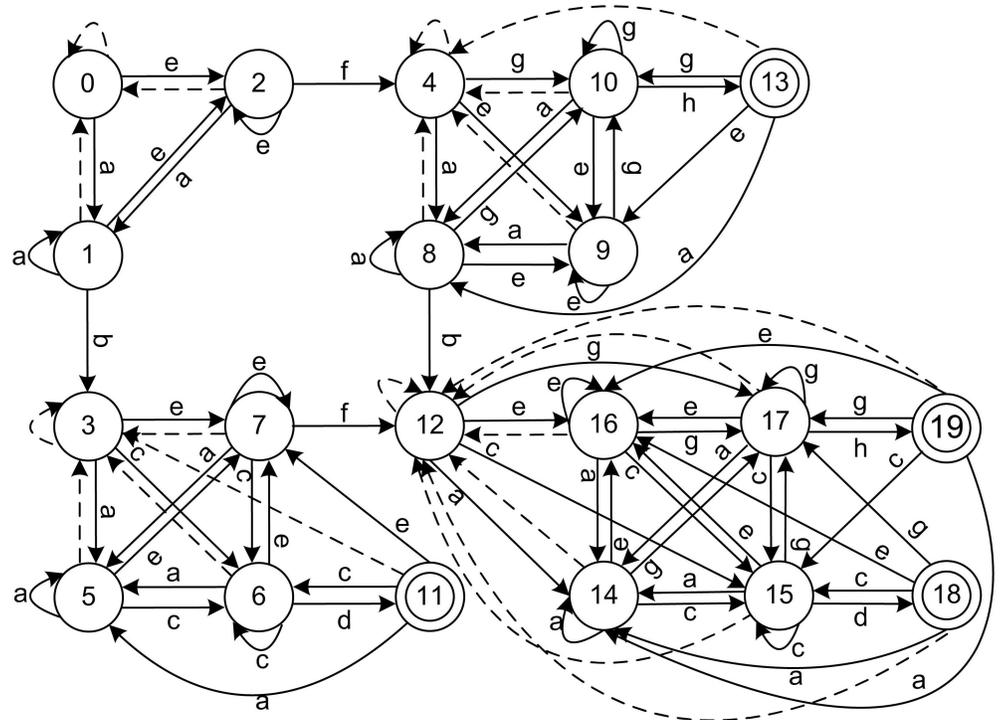


Fig 1. Original DFA state transition digraph for regular expressions $(ab \cdot *cd, ef \cdot *gh)$. The numerical circle represents states in DFA, and the state with double circles represents it is an accepted state. The direct edges with a symbol represents transitions in DFA, and the dashed direct edges represents transitions transfer to a common state with symbols that have not been shown.

doi:10.1371/journal.pone.0165864.g001

accepted states, and s_i and s_j are equivalent when they are accepted states, then states p and q are said to be equivalent, which denoted as $p \equiv q$; otherwise, p and p are distinguishable ($p \not\equiv q$).

According to the above, the minimization of a DFA can be treated as a problem of determining whether any two states in the state set of the DFA are distinguishable or equivalent. The most straightforward approach to this problem is to examine the consistency of strings that reach accepted states from any two states, that is, verifying $L_p(w) \equiv L_q(w)$ directly, where $L_p(w) = \{s | \hat{\delta}(p, w) = s\}$. However, the cost of doing so is great. For a cyclic automaton, it is impossible to enumerate all strings that can go from a certain state to an accepted state. Fortunately, the equivalence of any two states can be determined by the relationship between states or transitions. In this paper, the idea of a partition is adopted to obtain the minimal DFA. All states in the DFA state set are considered as a whole, and the nonequivalent states are found step by step. The minimal DFA is obtained when no equivalent states exist. To explain the algorithm more clearly, we define a partition as follows.

Partition. A partition of set E is a family P of nonempty, pairwise, disjoint subsets of E such that $E = \bigcup_{P_k \in P} P_k$. $|P|$ represents the number of subsets in the partition P . If $|P|$ is greater, the partition is more refined.

To partition the state set efficiently, the brute force method is discarded, and a series of propositions for distinguishing states are introduced.

Proposition 1. For states $p \in Q, q \in Q$, if there exists a symbol a that maintains $\delta(p, a) = s$ and $\delta(q, a) = t$ and furthermore, s and t are distinguishable, then p and q are distinguishable.

Proposition 1 proves that the partition can be obtained according to the transitions in the DFA; however, it is an inefficient way to obtain the minimal DFA directly. To improve the

Table 1. State transition table for regular expressions (ab.*cd, ef.*gh).

Σ	a	b	c	d	e	f	g	h
S								
0	1	0	0	0	2	0	0	0
1	1	3	0	0	2	0	0	0
2	1	0	0	0	2	4	0	0
3	5	3	6	3	7	3	3	3
4	8	4	4	4	9	4	10	4
5	5	3	6	3	7	3	3	3
6	5	3	6	11	7	3	3	3
7	5	3	6	3	7	12	3	3
8	8	12	4	4	9	4	10	4
9	8	4	4	4	9	4	10	4
10	8	4	4	4	9	4	10	13
11	5	3	6	3	7	3	3	3
12	14	12	15	12	16	12	17	12
13	8	4	4	4	9	4	10	4
14	14	12	15	12	16	12	17	12
15	14	12	15	18	16	12	17	12
16	14	12	15	12	16	12	17	12
17	14	12	15	12	16	12	17	19
18	14	12	15	12	16	12	17	12
19	14	12	15	12	16	12	17	12

doi:10.1371/journal.pone.0165864.t001

efficiency of the minimization, the backward depth is defined and its related propositions are introduced.

Backward Depth. The backward depth from state p to accepted state t is defined by the length of the shortest path from p to t , and can be formally denoted as

$$BD(p, t) = \min \{ |w| \mid \hat{\delta}(p, w) = t, t \in F \}.$$

In other words, the backward depth is the short length of words consumed in the process of state p transfer to accepted state t . The backward depth information for the DFA can be constructed using the reversal DFA and algorithm proposed in the next section. The reversal DFA is defined as follows.

Reversal DFA. Reversal DFA R of DFA D is also represented using a five-tuple $R = \langle Q, \Sigma, \delta^-, q_0, F \rangle$. The unique difference between R and D is the reversal transition function $\delta^-: Q \times \Sigma \rightarrow Q^*$, where $\delta^-(p, a) = \{q \mid \delta(q, a) = p\}$, and Q^* is the power set of Q .

The backward depth information can be used to coarsely partition the states according to the following proposition.

Proposition 2. If the backward depths of two states p and q for any accepted state t are different, p and q must be distinguishable. Formally, if $BD(p, t) \neq BD(q, t)$, then $p \neq q$.

It is possible that the DFA has many accepted states, so we can obtain the backward depth information of one state to many different accepted states. The state set can be partitioned according to the backward depth information for each accepted state; thus, many partitions are generated. We can refine a partition using the backward depth information according the following proposition.

Proposition 3. Given two partitions U and V of set E , we say that $W = U \cap V$ refines U and V . This means if U_i and V_j are elements of U and V , respectively, then these two elements will be divided into three elements in W , i.e., $W_k = U_i \cap V_j$, $W_m = U_i \setminus W_k$, and $W_n = V_j \setminus W_k$.

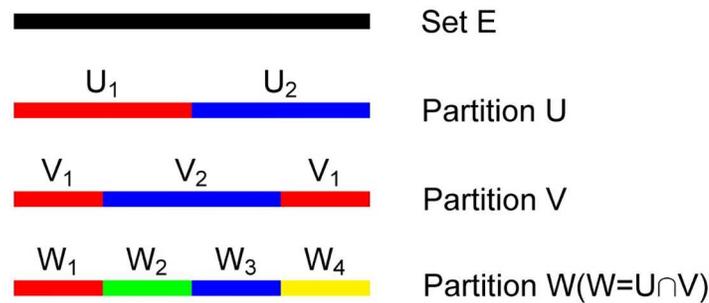


Fig 2. Example of obtaining a refined partition from coarse partitions. U, V and W represent different partitions for the set E . The bar with different colors represents a subset obtained by the corresponding partition, and the W partition is generated by the intersection of U and V .

doi:10.1371/journal.pone.0165864.g002

To explain it clearly, an example of proposition 3 is given in Fig 2. Different colors represent different elements in the corresponding partitions. Partition U includes two elements represented by red and blue segments, and V also has two parts, colored red and blue. The result W is divided into four segments colored red, blue, green, and yellow.

Although a comparatively refined partition of a state set can be obtained using backward depth information, it may not be the final minimal DFA. To obtain the final minimal DFA, refinement using a hash table is presented in the next section.

Minimal DFA Construction Algorithm

The minimization algorithm proposed in this paper solves the task in two main phases. In the first phase, the backward depth information for every accepted state is generated for each state, and the state set is grouped into many blocks. In the second phase, states in the same block are refined according to the hash table going from the deepest to the shallowest state.

Coarse Partitioning by Backward Depth Information

The task of coarse partitioning is to partition the state set using backward depth information, and then divide it into many blocks. A block is a subset of state set Q , and is gotten by the partition operation according to backward depth information. States in one block are not distinguishable according to the backward depth information after coarse partitioning, so they are instead refined using the hash table presented in the next subsection. There are two steps in coarse partitioning: building backward depth information and partitioning the state set using this information according to proposition 3.

Instead of the maximum level used by Revuz’s algorithm [22], backward depth information is used to initially partition the state set. The maximum level can only be obtained in acyclic automata, which constrains the application range of corresponding minimization algorithms. The time complexity of Revuz’s algorithm is linear because its target is an acyclic automaton, whose underlying graph is a tree. The algorithm in this paper can run in linear time when the automata is acyclic, and can be used in automata with more high topological complexity.

The pseudo-code of building backward depth information is shown in Fig 3. When building backward depth information, the reversal DFA is first constructed (lines 4–9). Then, the reversal DFA is traversed in a breadth-first fashion from one certain accepted state t using a lexicographic sequence to obtain a coarse partition consisting of blocks. The least number of symbols consumed from state s to accepted state t is the backward depth of s to t , denoted as $BD(s, t)$ (lines 10–25). To use backward depth information adequately, the symbols in the shortest path

Algorithm 1 Algorithm for building backward depth information

Input: DFA D

Output: The backward depth information of D

```

1: list  $\delta^-[[Q]][[\Sigma]]$ 
2: list queue
3: set  $BD[[Q]][[F]]$ 
4: for all  $c \in \Sigma$  do
5:   for all  $s \in Q$  do
6:      $t \leftarrow \delta(s, c)$ 
7:      $\delta^-[t][c].push(s)$ 
8:   end for
9: end for
10: for all  $q \in F$  do
11:   queue.push(q)
12:   while !queue.empty() do
13:      $s \leftarrow queue.pop()$ 
14:     for all  $c \in \Sigma$  do
15:       while  $\delta^-(s, c).notempty()$  do
16:          $p \leftarrow \delta^-(s, c).pop()$ 
17:         if  $BD[p][q].depth = NULL$  then
18:            $BD[p][q].depth \leftarrow BD[s][q] + 1$ 
19:            $BD[p][q].symbol \leftarrow c$ 
20:           queue.push(p)
21:         end if
22:       end while
23:     end for
24:   end while
25: end for

```

Fig 3. Algorithm for building backward depth information.

doi:10.1371/journal.pone.0165864.g003

to the state are saved, and then the states that have the same backward depth information are sorted into one block. Thus, the backward depth information is composed by two parts, the backward depth and the last symbol in the shortest path.

To explain the backward depth information intuitively, the backward depth information for accepted state 19 in the example in Table 1 is shown in Fig 4. The number above the dashed line represents the backward depth to state 19 for the states between the dashed lines, and the states in the same dashed box have the same backward depth information. The backward depth of state 19 is zero, because it consumes no character to reach itself. The states that have a backward depth of four are partitioned into four blocks. This occurs because block generation is determined by the backward depth, the symbol consumed to get to the next state in the graph, and whether the state is an accepted state or not. Block generation based on the backward depth information of an accepted state can be considered to be the shortest path tree whose root is the accepted state obtained from the state transition digraph.

The results of building the backward depth information for the example in Fig 4 are shown in Table 2. After building the backward depth information, we obtain the blocks for different accepted states, and these are shown in the corresponding column for the accepted state. The blocks can be further partitioned according to proposition 3, and the final blocks are shown in the last column in Table 2. Consider states p and q : if the backward depth information falls within different blocks for any accepted state, they will be arranged into different blocks.

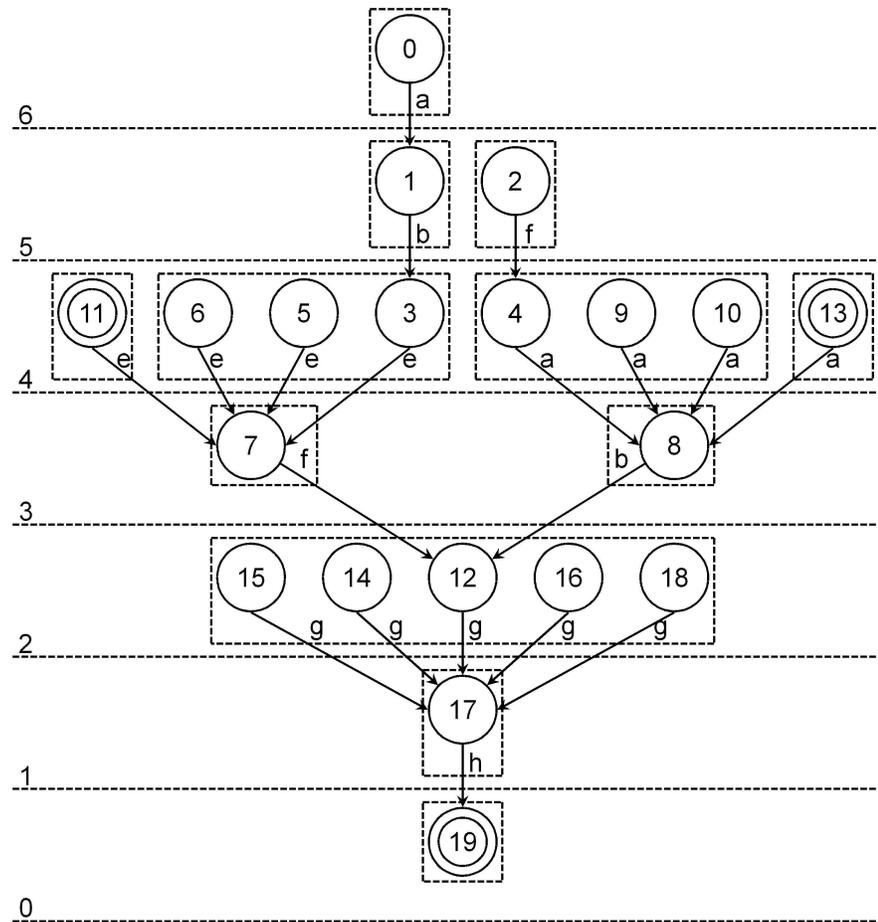


Fig 4. Backward depth information for accepted state “19” in Table 1. The number above the dashed line represents the backward depth, and the states in the same dashed box are in one block. The symbol near the transition is the character consumed in building the shortest path from the state transition graph, and is one part of the backward depth information.

doi:10.1371/journal.pone.0165864.g004

The coarse partition can be obtained by further block partitioning, and it can be transformed into a comparison of the backward depth information for different states. There are many methods for comparing backward depth information, and the simplest way is direct comparison. However, a direct comparison is costly with respect to time. In this paper, we calculate the hash value (HV) for each state according to the backward information, and then build the hash table. The final blocks can be obtained according to the hash table, that is, if the HVs of two states p and q are equal, they will be located in the same hash table position (HTP), and then they are assigned to the same block, and vice versa. The pseudo-code of coarse partition algorithm based on backward depth information is shown in Fig 5. The backward depth information of state s is denoted as $BDI(s)$, and it is composed by the connection of the backward depth to different accepted states (line 6–8). The hash values and the their positions in the hash table of backward depth information for each state are computed in line 9–11, and then the coarse partition is generated by the hash table (line 13–19).

In Table 2, blocks are created according to the backward depth information for an accepted state, and the final blocks created according to all backward depth information are listed in the last column. The final block partitioning in the last column may be the final minimization. In other words, the minimal DFA may be found without further computation.

Table 2. Backward depth information for automata with the regular expressions (ab.*cd, ef.*gh).

States	Accepted	11		13		18		19		Final block
		BD	Block	BD	Block	BD	Block	BD	Block	
0	0	4,a	0	4,e	0	6,e	0	6,e	0	0
1	0	3,b	1	4,e	0	5,b	1	5,b	1	1
2	0	4,a	0	3,f	1	5,f	2	5,f	2	2
3	0	2,c	2	-1	2	4,e	3	4,e	3	3
4	0	-1	3	2,g	3	4,a	4	4,a	4	4
5	0	2,c	2	-1	2	4,e	3	4,e	3	3
6	0	1,d	4	-1	2	4,e	3	4,e	3	5
7	0	2,c	2	-1	2	3,f	5	3,f	5	6
8	0	-1	3	2,g	3	3,b	6	3,b	6	7
9	0	-1	3	2,g	3	4,a	4	4,a	4	4
10	0	-1	3	1,h	4	4,a	4	4,a	4	8
11	1	0	5	-1	5	4,e	7	4,e	7	9
12	0	-1	3	-1	2	2,c	8	2,g	8	10
13	1	-1	6	-1	6	4,a	9	4,g	9	11
14	0	-1	3	-1	2	2,c	8	2,g	8	10
15	0	-1	3	-1	2	1,d	10	2,g	8	12
16	0	-1	3	-1	2	2,c	8	2,g	8	10
17	0	-1	3	-1	2	2,c	8	1,h	10	13
18	1	-1	7	-1	5	0	11	2,g	11	14
19	1	-1	6	-1	7	2,c	12	0	12	15

doi:10.1371/journal.pone.0165864.t002

Algorithm 2 Coarse partition algorithm based on backward depth information

Input: The backward depth information of D

Output: The coarse partition based on backward depth information

```

1: unsigned long HV
2: unsigned long HTP
3: list HashTable
4: block B
5: for all  $s \in Q$  do
6:   for all  $q \in F$  do
7:      $BDI(s) = strcat(BDI(s), BD(s, q))$ 
8:   end for
9:    $HV = Hash(BDI(s))$ 
10:   $HTP = HV / |Q|$ 
11:   $HashTable[HTP].push(s)$ 
12: end for
13: for all  $i < HashTable.size()$  do
14:   if  $HashTable[i].empty()$  then
15:      $B[k] \leftarrow HashTable[i]$ 
16:      $k++$ 
17:   end if
18:    $i++$ 
19: end for

```

Fig 5. Coarse partition algorithm based on backward depth information.

doi:10.1371/journal.pone.0165864.g005

Refinement Based on Hash Table

When partitioning using backward depth information, not all information about the automata is used. Any two states that belong to different blocks are demonstrably distinguishable, however, it is possible that states in the same block are not equivalent. Therefore, the final minimal DFA can sometimes not be obtained by the first step. To finish the minimization, we use the transition information to refine the blocks in the coarse partition. That is to say, states in one block may be partitioned into new blocks by a hash table of the transition information.

A state in a DFA has many transitions to the next states according to the symbols in the symbol set. A transition is composed of the starting state, symbol, and the ending state, as illustrated in Fig 1. The transitions that have the same starting state are called the state's transition set. If the transition set of any two states in the same block are different, then the two states are deterministically distinguishable. The refinement of the coarse partition is obtained by a comparison to the transition sets using a hash table. For symbol a , if two states p and q in one block are transferred to different blocks, then p and q must be distinguishable. The correctness of this approach is proved by proposition 1.

For blocks that have only one state, it is unnecessary to refine them further. Because blocks are coadjacent according to their transitions, many iterations of refinement could be needed for blocks that have many states. Refinement is terminated when no new blocks are generated. If the original DFA is minimal or acyclic, no iterations are needed. Details of the refinement are given in the following.

Suppose the cardinality of block i is n_i . The time complexity of directly comparing any two states in the block is then $O(n_i^2)$. Instead of directly comparing transition sets, we establish a hash table for every block and use it to perform refinements. The pseudo-code for refinement using a hash table is given in Fig 6. The hash value of state p is denoted as $HV(p)$, and the input of the hash calculation is the transition information of the state, which can be expressed as $HV(p) = Hash(BDI^-(p))$ (line 7). The transition information of the state is computed in line 4–6, where $B^-(\delta(p, c))$ represents the information of the block that includes element $\delta(p, c)$ (lines 5–9). The position of a state in the corresponding block's hash table can be obtained by its HV, which is represented as $HTP(s)$ (line 8). If the positions of the different states are the same, we consider the two states to be equivalent; otherwise, they are distinguishable. If states in a block are distinguishable, the block is divided into multiple blocks (lines 11–20). If a block has only one state, the calculations are ignored. If the number of blocks does not increase during refinement, the algorithm terminates.

Table 3 shows the results of hash table refinement using the example in Table 1. To explain hash table refinement properly, the coarse partition for accepted state 19 instead of the final coarse partition shown in the last subsection is used. In the first refinement, block 3, which includes states 3, 5, and 6, is separated into blocks 3 (states 3 and 5) and 13 (state 6) because the position of state 6 is not the same as that of states 3 and 5 in the hash table. Similarly, state 14 is separated from block 4, and added to a new block 14. State 15 is moved from block 8 to the new block 15. In the second refinement, no new blocks are generated because there are no states in the same block that have different hash table positions, so the refinement is terminated.

The advantage of the hash table is that it has lower time complexity: $O(n)$ compared to the $O(n^2)$ of direct comparison. The other noteworthy problem is hash table collision, which occurs when two inputs have the same position although they are different. Distinguishable states will not be found if hash collision occurs among the transition information of these states. In order to guarantee the correctness of the minimal algorithm, an inspection of hash collisions is necessary. This means that the state transition information in the same block must

Algorithm 3 Refinement algorithm based on Hash table

Input: The coarse partition of DFA D
Output: Minimum DFA for DFA D

```

1: Bool  $FstEle \leftarrow True$ 
2: for all  $B$  do
3:   for all  $s \in B[i]$  do
4:     for all  $c \in \Sigma$  do
5:        $BDI^-(s) = strcat(BDI^-(s), B^-(\delta(s, c)))$ 
6:     end for
7:      $HV(s) = Hash(BDI^-(s))$ 
8:      $HTP = HV(s)/|Q|$ 
9:      $HashTable[HTP].push(s)$ 
10:   end for
11:   if  $HashTable[j] \neq NULL$  then
12:     if  $FstEle = True$  then
13:        $B[i] \leftarrow HashTable[j]$ 
14:        $FstEle \leftarrow False$ 
15:     else
16:        $B[|B| + 1] \leftarrow HashTable[j]$ 
17:     end if
18:   end if
19:    $HashTable.Clear()$ 
20:    $FstEle \leftarrow True$ 
21: end for

```

Fig 6. Refinement algorithm based on hash table.

doi:10.1371/journal.pone.0165864.g006

Table 3. Hash table refinement based on the coarse partition of accepted state 19.

States	19		HV(Fst)	HTP(Fst)	Block(Fst)	HV(Sec)	HTP(Sec)	Final Block
	BD	Block						
0	6,e	0			0			0
1	5,b	1			1			1
2	5,f	2			2			2
3	4,e	3	701180389	7	3	3783612131	5	3
4	4,a	4	2287599808	4	4	233390966	11	4
5	4,e	3	701180389	7	3	3783612131	5	3
6	4,e	3	932010536	2	13			5
7	3,f	5			5			6
8	3,b	6			6			7
9	4,a	4	2287599808	4	4	233390966	11	4
10	4,a	4	285902165	17	14			8
11	4,e	7			7			9
12	2,g	8	248919426	0	8	102449418	15	10
13	4,a	9			9			11
14	2,g	8	248919426	0	8	102449418	15	10
15	2,g	8	59348337	6	15			12
16	2,g	8	248919426	0	8	102449418	15	10
17	1,h	10			10			13
18	2,g	11			11			14
19	0	12			12			15

doi:10.1371/journal.pone.0165864.t003

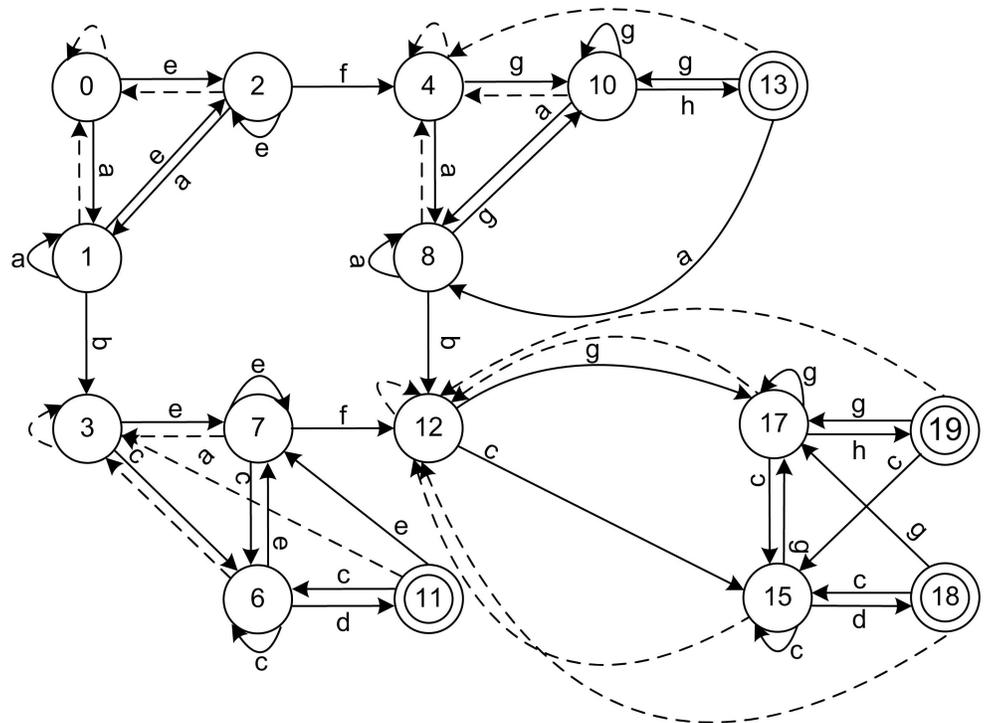


Fig 7. Minimal DFA using the algorithm based on backward depth information for regular expressions (ab.*cd, ef.*gh). The states 4, 9, 12 and 14 are disappeared for the reason that these have been merged into different states, respectively. The minimal DFA is equivalent with original DFA functionally, and is isomorphic with the final block within Table 3.

doi:10.1371/journal.pone.0165864.g007

be directly compared once more. The inspections are then performed after the refinements have been terminated, reducing the number of inspections.

To maintain the efficiency of the minimal algorithm, hash collisions should occur with low probability. The frequency of hash collisions is influenced primarily by the hash function and size of the hash table. An exclusive-or hash function [30, 31] is selected for the hash calculations in this paper, and the size of the hash table is the number of states in the DFA. Because the hash table is large enough and an exclusive-or hash function has perfect performance, few collisions occur. Usually, there are no collisions. Even when collisions occur, few inspections are needed because only those states who have the same position in the hash table need to be recalculated.

The final algorithm mainly consists of the backward depth information-based coarse partitioning and hash table refinement. The minimal DFA for the above-mentioned example is shown in Fig 7. As shown in the figure, states 3 and 5 in the original DFA are equivalent and merged into state 3 in the minimal DFA, states 4 and 9 are merged into state 4, and states 12, 14, and 16 are merged into state 12 in the final minimization of the DFA.

Experiments and Results

To evaluate the performance of the algorithms in this paper, Hopcroft’s algorithm, which known as the most efficient DFA minimization algorithm and has been utilized for various DFAs, was selected for comparison. The proposed algorithm and Hopcroft’s algorithm were implemented in C++ and run under GNU/Linux on a computer with an Intel processor Core i5 (3.2GHz) and 4 GB DDR3 (800 MHz). In order comprehensively and meaningfully compare

the two algorithms, three different classes of automata with increasing topological complexity were tested.

At present, there is no standard automata test set for evaluating DFA minimization algorithms. The automata used in this experiment were generated by a workload by Becchi [32], which mainly consists of regular expression generation according to some parameters. An NFA was produced starting from regular expressions using the Thompson method [10], and a DFA was generated by subset construction [12]. Because the topological complexity is mainly determined by wildcard and dot-star terms in regular expression sets, we fix all parameters except for the frequency of wildcard and dot-star terms. The other parameters were set to their default values in the experiment, and the input symbols set Σ was the ASCII alphabet.

The experimental results are shown in Fig 8. In this graph, the horizontal axis represents the number of states in the original DFAs, and the vertical axis indicates the time consumed in microseconds. Fig 8(a) and 8(b) show the results for automata with the lowest topological complexity, which do not have any wildcards or dot-star terms in the regular expressions. In this case, the algorithm presented in this paper performs better. This is because the asymptotic time complexity is linear when the automata degrades to an acyclic automata. The results for the automata with medium topological complexity (1% wildcard and dot-star terms) are shown in Fig 8(c). Here, the proposed algorithm does not perform better than Hopcroft's algorithm. The reason for this is that the proposed algorithm encounters an imperfect partition based on the backward depth information. As shown in Fig 8(d), the proposed algorithm achieves a better time complexity than Hopcroft's algorithm for the automata with high topological complexity (3% wildcard and dot-star terms).

The time complexity of the proposed algorithm is $O(n + \sum(n_i) + k)$, and is composed of three main aspects. First, the time complexity of the coarse partition is $O(n)$, where n represents the number of states in the original DFA. To obtain the backward depth information, the only operation is the construction of the reversal DFA, which consumes n periods. The time complexity of the refinements on block i without considering hash collisions is $O(n_i)$, and there may be many blocks that need to be refined by the hash table. Thus, the total time complexity of the hash table refinements without considering hash collisions is $O(\sum(n_i))$. The hash collision inspections consume fewer cycles because there are not many of them. The time complexity of hash collision inspections is $O(k)$, where k is a random number. The hash table refinements account for the highest proportion of the time complexity because they may be iterated many times. The number of refinement iterations is mainly influenced by the DFA's topological complexity. If the DFA has an underlying graph with low topological complexity, the states are partitioned primarily by the coarse partition, and the algorithm proposed in this paper will consume comparatively few cycles. For the DFA with high topological complexity, although the state set cannot be partitioned in the coarse partitioning, it is partitioned well in the early stages of the hash table refinements.

Discussion and Conclusions

The experiments show that the proposed algorithm achieves better time complexity in most cases. This is for three main reasons: the mechanism of coarse partitioning according to backward depth information, the highly efficient comparison of transition information using a hash table, and few hash collisions. The backward depth information that is used to obtain the coarse partition is generated when the reversal DFA is constructed. In the partition refinement to obtain the minimal DFA, the transition information is compared by hash table instead of directly, which decreases the time complexity from $O(n_i^2)$ to $O(n_i)$.

In addition to the strong quantitative performance results, the proposed method has substantial qualitative advantages, including greater generality and simplicity. Compared to

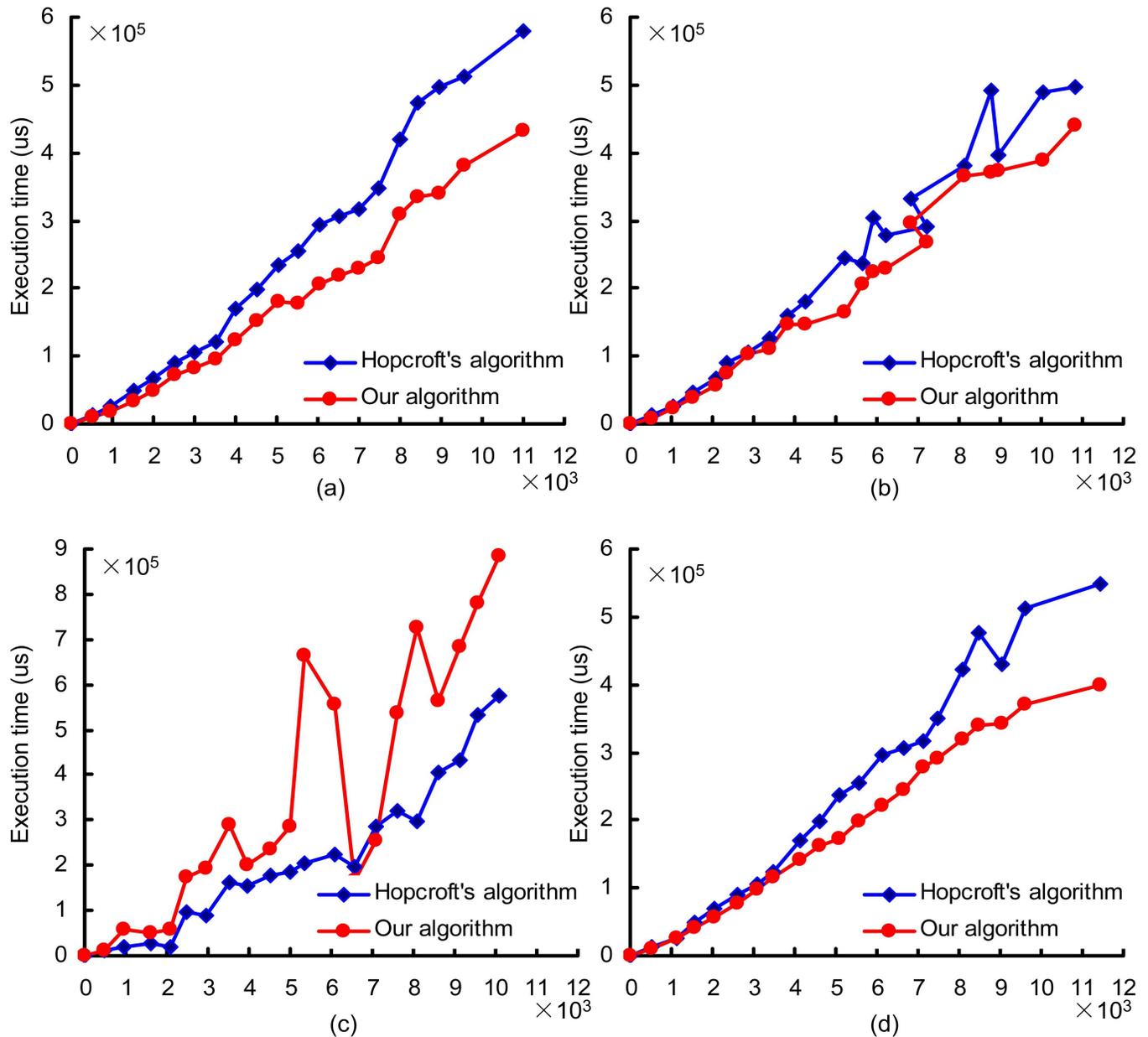


Fig 8. Experimental results for automata with different topological complexities. The horizontal axis represents the number of states in the original DFAs, and the vertical axis indicates the time consumed in microseconds. (a) DFA with an underlying graph consisting of a dictionary, (b) DFA with an underlying graph consisting of a tree, (c) DFA with 1% wildcards and dot-star terms, and (d) DFA with 3% wildcards and dot-star terms.

doi:10.1371/journal.pone.0165864.g008

previous minimization algorithms, the backward depth information can be obtained for any type of DFA. Thus, in contrast to previous work, the algorithm proposed in this paper can be used not only on acyclic automata and automata with simple cycles, but also on topologically complicated automata. The maximum level proposed in Revuz's algorithm is the longest path from the state to the final state, and if the underlying graph has circles, then the maximum level cannot be obtained. All other algorithms except for Hopcroft's algorithm have similar disadvantages with respect to DFA minimization generality. Furthermore, the algorithm proposed in this paper can be extended to other types of automata; for example, incomplete DFAs. In an

incomplete DFA, states may have no transitions for some symbols; however, this has no influence on the implementation of the algorithm. The main operations consist of the building of backward depth information and hash table refinement, and these operate without considering the integrality of the transitions.

In summary, we propose a DFA minimization algorithm based on backward depth information in this paper that is both simple and has better time complexity and greater generality than previous approaches.

Supporting Information

S1 File. The proofs of propositions. S1 File includes Appendixes A and B. Appendix A describes the proof of Proposition 1, and Appendix B gives the proof of Proposition 2. (PDF)

Acknowledgments

The authors would like to thank the Laboratory of Network Test Technology at the National University of Defense Technology for their helpful comments and discussions. We would like to thank Editage (www.editage.com) for English language editing.

Author Contributions

Conceptualization: DL ZH YZ.

Data curation: DL YZ.

Formal analysis: DL ZH SS.

Investigation: DL ZH SS.

Methodology: DL YZ XG.

Project administration: ZH.

Resources: ZH SS.

Software: DL YZ.

Supervision: ZH SS.

Validation: DL XG.

Visualization: DL YZ.

Writing – original draft: DL ZH XG.

Writing – review & editing: DL.

References

1. Chomsky N, Miller GA. Finite state languages. *Information and Control*. 1958; 1(2):91–112. doi: [10.1016/S0019-9958\(58\)90082-2](https://doi.org/10.1016/S0019-9958(58)90082-2)
2. Biegler F, Daley M, Holzer M, McQuillan I. On the uniqueness of shuffle on words and finite languages. *Theoretical Computer Science*. 2009; 410(5):3711–3724. doi: [10.1016/j.tcs.2009.04.016](https://doi.org/10.1016/j.tcs.2009.04.016)
3. Aho AV, Sethi R, Ullman JD. *Compilers, principles, techniques, and tools*. Boston, MA, USA: Addison-Wesley Publishing Corporation; 1986.
4. Sun GQ, Jusup M, Zhen J, Yi W, Zhen W. Pattern transitions in spatial epidemics: Mechanisms and emergent properties. *Physics of Life Reviews*. 2016; doi: [10.1016/j.plrev.2016.08.002](https://doi.org/10.1016/j.plrev.2016.08.002)

5. Brodie BC, Cytron RK, Taylor DE. A Scalable Architecture For High-Throughput Regular-Expression Pattern Matching. *ACM SIGARCH computer architecture news*. 2006; 34:191–202. doi: [10.1145/1150019.1136500](https://doi.org/10.1145/1150019.1136500)
6. Clark CR, Schimmel DE. Efficient Reconfigurable Logic Circuits for Matching Complex Network Intrusion Detection Patterns. *Lecture Notes in Computer Science*. 2003; 2778:956–959. doi: [10.1007/978-3-540-45234-8_94](https://doi.org/10.1007/978-3-540-45234-8_94)
7. Hyunjin K, Kang-II C, Sang-II C. A memory-efficient deterministic finite automaton-based bit-split string matching scheme using pattern uniqueness in deep packet inspection. *PLoS One*. 2015; 10(5):286–293. doi: [10.1371/journal.pone.0126517](https://doi.org/10.1371/journal.pone.0126517) PMID: [25938779](https://pubmed.ncbi.nlm.nih.gov/25938779/)
8. Sun GQ, Wang SL, Ren Q, Jin Z, Wu YP. Effects of time delay and space on herbivore dynamics: linking inducible defenses of plants to herbivore outbreak. *Scientific Reports*. 2015; 5. doi: [10.1038/srep11246](https://doi.org/10.1038/srep11246) PMID: [26084812](https://pubmed.ncbi.nlm.nih.gov/26084812/)
9. Sun GQ, Wu ZY, Wang Z, Jin Z. Influence of isolation degree of spatial patterns on persistence of populations. *Nonlinear Dynamics*. 2016; 83:811–819. doi: [10.1007/s11071-015-2369-6](https://doi.org/10.1007/s11071-015-2369-6)
10. Thompson K. Regular expression search algorithm. *Communications of the ACM*. 1968; 11(6):419–422. doi: [10.1145/363347.363387](https://doi.org/10.1145/363347.363387)
11. Gluskov VM. The abstract theory of automata. *Russian Mathematical Surveys*. 1961; 16(5):3–62.
12. Hopcroft JE, Ullman JD. *Introduction to automata theory, languages and computation*. Boston, MA, USA: Addison-Wesley Publishing Corporation; 2007.
13. Hopcroft J. An nlogn algorithm for minimizing states in a finite automaton. *Theory of Machines and Computations*. 1971; p. 189–196. doi: [10.1016/B978-0-12-417750-5.50022-1](https://doi.org/10.1016/B978-0-12-417750-5.50022-1)
14. Berstel J, Carton O. On the complexity of Hopcroft's state minimization algorithm. In: *Implementation and Application of Automata, International Conference, CIAA 2004, Kingston, Canada, July 22–24*. vol. 3177. Springer-Verlag; 2004. p. 35–44.
15. Castiglione G, Restivo A, Sciortino M. Hopcroft's Algorithm and Cyclic Automata. In: *Martín-Vide C, Otto F, Fernau H, editors. Language and Automata Theory and Applications: Second International Conference, LATA 2008, Tarragona, Spain, March 13–19, 2008. Revised Papers*. Berlin, Heidelberg: Springer Berlin Heidelberg; 2008. p. 172–183.
16. Valmari A, Lehtinen P. Efficient minimization of DFAs with partial transition functions. *Eprint Arxiv*. 2008;.
17. Beal MP, Crochemore M. Minimizing local automata. In: *IEEE International Symposium on Information Theory; 2007*. p. 1376–1380.
18. Gries D. Describing an algorithm by Hopcroft. *Acta Informatica*. 1973; 2(2):97–109. doi: [10.1007/BF00264025](https://doi.org/10.1007/BF00264025)
19. Knuutila T. Re-describing an algorithm by Hopcroft. *Theoretical Computer Science*. 2001; 250(1–2):333–363. doi: [10.1016/S0304-3975\(99\)00150-4](https://doi.org/10.1016/S0304-3975(99)00150-4)
20. Moore EF. *Gedanken experiments on sequential machines*. Princeton University Press; 1956. doi: [10.1515/9781400882618-006](https://doi.org/10.1515/9781400882618-006)
21. David J. The average complexity of Moore's state minimization algorithm is $O(n \log \log n)$. *Lecture Notes in Computer Science*. 2010; 6281:318–329. doi: [10.1007/978-3-642-15155-2_29](https://doi.org/10.1007/978-3-642-15155-2_29)
22. Revuz D. Minimisation of acyclic deterministic automata in linear time. *Theoretical Computer Science*. 1992; 92(1):181–189. doi: [10.1016/0304-3975\(92\)90142-3](https://doi.org/10.1016/0304-3975(92)90142-3)
23. Almeida J, Zeitoun M. Description and analysis of a bottom-up DFA minimization algorithm. *Information Processing Letters*. 2008; 107(2):52–59. doi: [10.1016/j.ipl.2008.01.003](https://doi.org/10.1016/j.ipl.2008.01.003)
24. Watson BW. *Constructing minimal acyclic deterministic finite automata*. Springer Berlin Heidelberg; 2004.
25. Brzozowski JA. Canonical regular expressions and minimal state graphs for definite events. *Mathematical Theory of Automata*. 1962; 12(6):529–561.
26. Daciuk J. Comparison of construction algorithms for minimal, acyclic, deterministic, finite-state automata from sets of strings. In: *International Conference on Implementation and Application of Automata; 2002*. p. 255–261.
27. Lamperti G, Scandale M. Incremental determinization and minimization of finite acyclic automata. In: *IEEE International Conference on Systems, Man, and Cybernetics; 2013*. p. 2250–2257.
28. Carrasco RC, Forcada ML. Incremental construction and maintenance of minimal finite-state automata. *Computational Linguistics*. 2002; 28(2):207–216. doi: [10.1162/089120102760173652](https://doi.org/10.1162/089120102760173652)
29. Sakarovitch J. *Elements of automata theory*. New York, NY, USA: Cambridge University Press; 2009. doi: [10.1017/CBO9781139195218](https://doi.org/10.1017/CBO9781139195218)

30. Xiong B, Yang K, Li F, Chen X, Zhang J, Tang Q, et al. The impact of bitwise operators on hash uniformity in network packet processing. *International Journal of Communication Systems*. 2014; 27(11):3158–3184. doi: [10.1002/dac.2532](https://doi.org/10.1002/dac.2532)
31. Li L, Liu G, Pan Q. Extending second-preimage-resistant-like compress functions to XOR-based target collision resistant hash functions. In: *Information Management and Engineering (ICIME), 2010 The 2nd IEEE International Conference on*; 2010. p. 444–448.
32. Becchi M, Franklin M, Crowley P. A workload for evaluating deep packet inspection architectures. In: *Workload Characterization, 2008. IISWC 2008. IEEE International Symposium on*; 2008. p. 79–89.