**8th December 2021**

**CSCI 541 THEORY OF COMPUTING**

SHUBHAM DESHMUKH

(202006307 – x2020gfv)

**A SURVEY PAPER ON:**

**PROCEDURES FOR MINIMIZING FINITE AUTOMATA**

INSTRUCTED BY: DR. TAYLOR J. SMITH

DEPARTMENT OF COMPUTER SCIENCE

**St. FRANCIS XAVIER UNIVERSITY**

1. **Introduction**

Minimization of automata refers to the construction of finite automata with a minimum number of states which is equivalent to the original finite automata, the minimization process involves the removal of those states whose absence doesn’t affect the language acceptability of the automata. The number of states employed in a finite automaton is directly proportional to the size of the automata. As a result, it is empirical to limit the number of states and minimize the automata. Minimized automata do the same task with lesser states thus reducing the overall processing time its therefore an important issue in the use and implementation of high-level finite automata tools like image analysis, text processing, linguistic computer science, and other computer science applications, etc. Minimization can be broadly classified into two different types, The first is achieved through a series of refinements of a partition of the set of states, while the second is achieved through a series of state fusions or merges. This paper will discuss some of the minimization algorithms and techniques out of the two types, apart from the minimization of deterministic finite automata the report will also include minimization of non-deterministic automata and symbolic automata. Time complexity plays a major role in machine calculation and minimizing any automata results in saving time to solve the problem, the report will give a brief about the time complexity of some of the algorithms.

1. **Brzozowski’s algorithm**

Brzozowski's is a simple and easy to implement minimization algorithm and can apply to both deterministic and non-deterministic finite automata, Brzozowski's minimization algorithm is considerably different from the 2 kinds of iterative algorithms presented in this introduction of the report (by refinement and by fusion). However, its worst-case behavior is exponential but it is quite sufficient in some cases.

Given an automaton A = (Q, I, F, E) over an alphabet A, its reversal is the automaton denoted AR obtained by exchanging the initial and the final states, and by inverting the orientation of the edges. Formally AR = (Q, F, I, ER), where ER = {(p, a, q) | (q, a, p) ∈ E}. The basis for Brzozowski’s algorithm is the following result[4].

* 1. **Proposition**

If A is a finite deterministic automaton, and A∼ be the deterministic trim automaton obtained by determinizing and trimming the reversal AR. Then A∼ is minimal. See proof[10]

* 1. **Corollary**

if A is a finite automaton. Then (A∼) ∼ is the minimal automaton of A.

1. **Moore’s algorithm**

Moore’s minimization algorithm is only for deterministic finite automata and the minimization algorithm computes the Nerode equivalence by a stepwise refinement of some initial equivalence. It maintains a partition that begins by dividing the accepting and rejecting states and refines it until no further refinements are possible. The algorithm was given by Edward F. Moore. It is similar to Hopcroft’s Algorithm which will be discussed later in the report.

* 1. **Description**

Text

Description automatically generated

**Graphical user interface, text

Description automatically generated**

Fig. 3.1 Moore’s Minimization Algorithm [4]

The definitions of the Nerode equivalence and the Moore equivalence of order h can be rewritten using the partition notation described above. These are the equivalences that have been established.

Company name

Description automatically generated with low confidence

The Nerode equivalence and the Moore equivalence can be equal for the smallest value of h because the set of states is finite. This value of h is also called the depth of Moore’s algorithm and it only depends on the language recognized by the automata. It replaces the current partition with the coarsest common refinement of s + 1 partitions, one of which is the current partition and the others are the current partition's preimages under the transition functions for each of the input symbols, at each step. When this replacement does not modify the current partition, the algorithm ends.

We claim that automata with the greatest depth are slow, and more precisely, Moore automata are slow (l = n - 2 the worst case). These automata will be looked into later. We'll prove that they're equivalent to the slow Hopcroft automata.

* 1. **Complexity**

The worst-case time complexity is *O*(*n*2*s*). However, some research papers have studied different complexity of Moore’s algorithm and their results are as follows:

* the average time complexity of Moore’s state minimization algorithm is O(n log log n), where n is the number of states in the input automata and the number of letters in the alphabet is fixed. [12]
* Let K be a semi-automation with n states. The average complexity of Moore’s algorithm for the automata (K, F), for the uniform probability distribution over the sets F of final states, is O(n log n). [13]
* The average complexity of Moore’s algorithm, for the uniform probability overall complete automata with n states, is O(n log log n). [14]

1. **Hopcroft’s algorithm**

This minimization algorithm was given by Hopcroft and its for finding minimization of Deterministic automata. Its based on partition refinement, dividing DFA states into groups based on their behavior These are the equivalence classes of the Myhill–Nerode equivalence relation, which asserts that every two states of the same partition are equivalent if their behavior is the same for all input sequences. That is, the transitions defined by w should always move states a1 and a2 to equal states that both accept, or states that both reject, for every two states a1 and a2 that belong to the same equivalence class within the partition A, and every input word w. It should not be feasible for w to make a1 accept and a2 refuse, or vice versa.

**Text, letter

Description automatically generated**

Fig. 4.1 Hopcroft’s Minimization Algorithm [4]

* 1. **Description**

Hopcroft's algorithm is outlined above image. We denote by min(P, P′ ) the set of the smaller size of the two sets P and P ′, and any one of them if they have the same size. Let A being a deterministic automaton, the algorithm computes the coarsest congruence which saturates the set F of final states. It begins with the partition {F, Fc} which saturates F and refines it until it achieves congruence. These partition refinements are always achieved by breaking a class into two classes. The following is how the algorithm works. It keeps track of a current partition P = P1, P2,..., Pn, as well as a current set W of splitters, or pairings (W, a) that need to be processed, where W denotes a P class and a denotes a letter. The waiting set is designated by the letter W. When the waiting set W becomes empty, the algorithm grinds to a halt. The partition P is the coarsest congruence that saturates F when it comes to a halt. The partition F, Fc is the initial partition, and the starting set W comprises all pairings (min(F, Fc), a) for an A. The algorithm's main loop removes one splitter (W, a) from the waiting set W and conducts the actions listed below. The current partition's class P (containing the class W) is checked to see if it is split by the pair (W, a). Nothing is done if (W, a) does not split P. If (W, a) divides P into P ′ and P ′′, for example, the class P is replaced in the partition P by P ′ and P ′′. If the pair (P, b) is already in W for each letter b, it is replaced in W by the two pairs (P ′, b) and (P ′′, b), otherwise only the pair (min(P ′, P′′), b) is added to W. Because it is not defined which pair (W, a) is taken from W to be processed at each iteration of the main loop, the procedure is not truly deterministic. This means that there are many different ways to run an algorithm for a particular automaton. All of them, it turns out, provide the correct state partitioning. Various executions, on the other hand, may result in different splitting sequences and running times.

Hopcroft's algorithm has a few degrees of freedom. The way the waiting set is expressed, in particular, may have an impact on the algorithm's performance. Some research showed that when the waiting set is implemented as a queue (LIFO), the worst-case complexity indicated in the case of de Bruijn words remains the same, however, this complexity is never reached when the waiting set is implemented as a stack (FIFO). Hopcroft's algorithm necessitates the completion of the automaton. This could be a significant disadvantage if the automaton just has a few transitions. If a dictionary is represented as a trie, the average number of edges per state for the French dictionary is roughly 1.5 (personal communication of Dominique Revuz).

* 1. **Complexity**

Hopcroft’s Algorithm’s time complexity of any execution is inside O(k n log n) where k is the cardinality of the alphabet and n is the number of states of the given automaton. Some results have shown the two incomplete automata adaptations of Hopcroft's algorithm, with running time O(m log n), where n is the number of states and m is the number of transitions.

1. **Minimization by Fusion**

This is another family of minimization where minimization is done by fusion of finite states, This is commonly used in computational linguistics to represent dictionaries in a space-efficient manner. Given deterministic automata A with Q as a set of states, the signature of a state p is the set of pairs (a, q) ∈ A × Q such that p · a = q, together with a Boolean value denoting whether p is final or not. Two states p and q are called mergeable if and only if they have the same signature. The fusion or merge of two mergeable states p and q consists in replacing p and q by a single state. The state obtained by the fusion of two mergeable states has the same signature. It is not always possible to minimize an automaton using a sequence of fusions of states with the same signature. Below topics are some of the important minimization algorithms by fusion of states.

1. **Local Automata**

This is a type of fusion minimizing and where given be M.-P. B´eal and M. Crochemore. The algorithm minimizes a special class of deterministic automata by a sequence of state mergings and these automata are called local automata. The algorithm is linear in particular when the automaton is already small and the algorithm does not require the automaton to be complete.

The automata under consideration have several unique characteristics. To begin with, every state is both beginning and final. They are also irreducible, which means that their underlying graph is highly connected. Lastly, they are local meaning that two distinct cycles carry different labels. This implies that a cycle's labels are basic words because otherwise, distinct traversals of the cycle with the same name exist. Because all states of an automaton are final, two states p and q are mergeable if and only if for all letters a ∈ A, p · a is defined if and only if q · a and, if this is the case, then p · a = q · a.

A picture containing necklet

Description automatically generated

Fig 6.1 Left one is local automata and the right one is not.[4]

* 1. **Proposition**

If an irreducible local automaton is not minimal, then at least two of its states are mergeable. The proposition is taken from[4].

The minimization algorithm assumes that the alphabet is totally ordered. It uses the notion of partial signature. First, with each state q is associated the signature σ(q) = {a1p1a2p2 · · · ampm}, where {(a1, p1), . . . ,(am, pm)} is the signature of q, and the sequence is ordered by increasing value of the letters. Since all states are final, the Boolean indicator reporting this is omitted. A partial signature is any prefix {a1p1a2p2 · · · aipi} of a signature. The first step consists in building a signature tree that represents the sets of states sharing a common partial signature. The root of the tree represents the set of all states, associated with the empty signature. A node representing the sets of states with a partial signature {a1p1a2p2 · · · aipi} is the parent of the nodes representing the sets of states with a partial signature {a1p1a2p2 · · · aipiai+1pi+1}. As a consequence, leaves represent full signatures. All states that correspond to a leaf are mergeable. When mergeable states are detected in the signature tree, they can be merged. Then the signature tree has to be updated, and this is the difficult part of the algorithm.

* 1. **Complexity**

The algorithm’s time complexity is O(min(m(n−r+ 1), m log n)), where m is the number of edges, n is the number of states, and r is the number of states of the minimized automaton.

1. **Bottom-Up & Revuz’s minimization**

Bottom-up minimization is the process of using a bottom-up traversal to minimize an acyclic automaton. The children of a node are treated before the node itself in this traversal. Equivalent states are discovered and merged during the traverse. It has the characteristic that the check for (Nerode) equivalence reduces to signature equality. The most important thing is to organize the states that are candidates so that this check may be completed quickly. The bottom-up traversal can be structured in a variety of ways, such as as a depth-first search with the order of traversal of the children dictated by the order of the edge labels. Another method of traversal is to increase height, as seen in Revuz's algorithm which is discussed below.

Text

Description automatically generated

Fig 7.1 Revuz’s Minimization Algorithm[4]

Revuz Algorithm was the first one to have an explicit explanation of a bottom-up minimization algorithm implementation in linear time. The length of the longest path starting at p is defined as the height of a state p in an acyclic automaton. It's also the length of the sub-automaton's longest word in the language at p. There is no difference in height between the two equivalent states. The algorithm that Revuz uses works by increasing the height of the object. A bottom-up traverse can be used to compute heights in linear time. This traversal collects lists of states of a particular height. If the edges starting in a state have been sorted, computing a state's signature is simple (by a bucket sort for instance to remain within the linear time constraint). A lexicographic sort is used to sort states by their signature yet again. The bottom-up traversal of the trie is used by Revuz's algorithm. This traversal is defined by raising the height of states, which simplifies the search for equivalent states. The algorithm could be organized differently if you consider another method for checking signatures, such as hash coding. For example, lexicographic order can be used instead of heights to traverse the tree. Another check could be used in place of the algorithm's final item. The hash code of any state that must be in the minimal automaton is recorded. When computing a state's signature, it's important to see if the hash code is valid. If it isn't already in the register, it is, and the hash code takes its place.

Diagram

Description automatically generated

Fig 7.2 Left one is a normal automaton, the right one is a minimized automaton by merging of states.

* 1. **Complexity**

The runtime complexity of Revuz’s Algorithm is O(m) for an automaton with m edges.

1. **The algorithm of Almeida and Zeitoun**

The Algorithm is given Almeida and Zeitoun and is an extension of the bottom-up minimization algorithm to automata that contain only simple cycles. We call it simple If every nontrivial strongly connected component is a simple cycle, that is if every vertex has exactly one successor vertex in this component. They are the simplest infinite regular languages. Simple automata are fascinating because they can recognize bounded regular languages or, in other words, languages with polynomial growth.

* 1. **Description**

The minimization process can be divided into two parts: minimization of an acyclic automaton and minimization of the set of strongly connected components. There are three subproblems: (1) minimization of each strongly connected component, (2) identification and fusion of isomorphic minimized strongly connected components, and (3) wrapping, which involves merging states that are equivalent to states in a strongly connected component but not in this component. If these subproblems can be solved in linear time, the entire automaton can be reduced in linear time using a bottom-up technique that is a refinement of Revuz's algorithm. As long as there are no nontrivial tightly connected components, the technique works like Revuz's algorithm. The cycle is small if and only if the word made up of the signature sequence is primitive. Similarly, to see if two (primitive) cycles may be merged, one looks at the weak signatures' words to see if they are conjugate.

**Graphical user interface, text

Description automatically generated**

Fig. 8.1 Almeida & Zeitoun Algorithm[4]

Diagram

Description automatically generated

Fig 8.2 Left One is a simple automaton, the right one is the minimized automata by merging states.

A picture containing clock

Description automatically generated

Fig 8.3 Left One is a simple automaton, the right one is minimized by wrapping & merging states.

* 1. **Complexity**

The minimization issue can be solved in the time complexity of O(f(d + l)), where d is the number of transitions, for any function f such that f(n)/n is nondecreasing, if and only if three subproblems have the same worst-case complexity.[15]

1. **Dynamic minimization**

It’s a different type of minimization where minimization is attained while insertion or deletion is done.

* 1. **Description**

Let A = (Q, i, T ) be the minimal automaton recognizing a language L over the alphabet A, and let w be a word in L. The minimum automaton that recognizes the complement A w is denoted by the letter A∗ \ w. There are n + 2 states in the automaton, with n equaling |w|. Among them, the set P of w prefixes is associated with n + 1 states. The final state, designated by, is a sinking state. The language L \ w is equal to L ∩ (A∗ \ w), so it is recognized by the trimmed part B of the product automaton A × Aw. Apart from the initial (i, ε), and its states are of three kinds. Intact states (q, ⊥) with q ∈ Q, cloned states (i · w, ⊥), useless states (i, ⊥) (removed while trimming the automata). The final step is to retain the basic automaton by renaming a state q as (q, ⊥), adding a cloned path, and removing useless states if it is no longer available, as well as the states that can only be reached from this state. There is another way of construction that’s, without relying on the presence of the automaton mentioned earlier. To do so, start with the new initial state (i, ε) and build only the parts of the product automaton that are accessible.

* 1. **Complexity**

When not taking advantage of existing automata the time complexity is O(n + |w|), where n is the number of states of the initial, On the other hand taking the existing automata into account gives time complexity of O(|w|).

1. **Non-Deterministic Automata Minimization**

Previously all the algorithms mentioned were minimization procedures for Deterministic finite automata, The effort of reducing a given nondeterministic finite automaton (NFA) into an equivalent DFA with the smallest number of states, transitions, or both is known as NFA minimization. DFA minimization has efficient methods, whereas NFA minimization is PSPACE-complete. NFA is not unique and If all automata that recognize the same language have the same amount of states, a nondeterministic automaton is minimal. There may be several NFAs of similar size that accepts the same regular language, but no corresponding NFA or DFA with fewer states. A minimization technique was given by Tsunehiko Kameda but due to the complexity of NFA minimization the minimized construction doesn’t always yield an equivalent NDA but he has shown that at least one of the NDA’s constructed is equivalent. [16] Given a finite automaton A, we may deduce from it A a  matrix of 1s and Os, known as a reduced automaton matrix RAM) of A, and prove that each state of A corresponds to a grid across the RAM in a specific fashion. A grid is made up of a collection of rows and columns in a RAM where only l's appear at intersections. It's also demonstrated that the union of all the grids, each of which corresponds to a state of A, covers all of a RAM's 1 entries. We then "synthesize" a minimum state NDA by reversing this analysis. We start by creating a RAM for a finite automaton, and then we look for a set of grids that covers all and only its 1 entry. We create an NDA whose states correlate to the grids in the set using the intersection rule provided in the text.

1. **Symbolic Automata**

Symbolic Automata are a type of automaton that uses symbolic alphabets rather than finite ones. The majority of classical automata algorithms rely on a finite alphabet, therefore generalizing them to a symbolic environment is not an easy operation. Minimization of symbolic automata can be done using the classic minimization algorithms like Huffman-Moore’s and Hopcroft’s algorithms and define the properties of minimality of symbolic automata. But the classic algorithm is blown up due to the predicate refinement. Hence a minimization without minterm generation has been performed The idea behind Min SFA is that it is not required to supply the exact witness that defines the split when dividing a partition's part; instead, it is sufficient to check if some witness (or witness set) exists. The entire proof and result can be read at [3].

1. **Summary**

Minimization is important because it reduces identical operations, minimising a finite automaton is highly valuable in making compilers run quicker, increasing the overall compilation time, and increasing the program's processing performance. Running a program that is a second or two faster than other programs can greatly expand the program's scope this is where minimization can come in handy. The report discussed some of the important minimization algorithms, their time complexity, and different ways in which the minimization process was done.

**References**

[1] Watson, Bruce W. A taxonomy of finite automata minimization algorithms. The Eindhoven University of Technology, Department of Mathematics and Computing Science, Computing Science Section, 1993. URL [download (psu.edu)](https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.462.87&rep=rep1&type=pdf)

[2] Minimization of finite automata by Dr. K.R. Chowdhary. [Minimization of Finite Automata (krchowdhary.com)](http://krchowdhary.com/toc/5-minimize.pdf)

[3] Loris D'Antoni and Margus Veanes. 2014. Minimization of symbolic automata. <i>SIGPLAN Not.</i> 49, 1 (January 2014), 541–553. DOI:https://doi.org/10.1145/2578855.2535849

[4] Berstel, Jean, et al. "Minimization of automata." *arXiv preprint arXiv:1010.5318* (2010). URL [arXiv:1010.5318v3 [cs.FL] 30 Dec 2010](https://arxiv.org/pdf/1010.5318.pdf)

[5] Nelson, R. (1955). D. A. Huffman. The synthesis of sequential switching circuits. Journal of the Franklin Institute, vol. 257 (1954), pp. 161–190, 275–303. *Journal of Symbolic Logic,* *20*(1), 69-70. doi:10.2307/2268078

[6] . Moore. Gedanken-experiments on sequential machines. Automata Studies, Annals of mathematics studies, 1956.

[7] . Hopcroft. An nlogn algorithm for minimizing states in a finite automaton. In Z. Kohavi, editor, Theory of machines and computations, Proc. Internat. Sympos., Technion, Haifa, New York, 1971. Academic Press.

[8] Thomas Colcombet and Daniela Petrişan. 2017. Automata and minimization. ACM SIGLOG News 4, 2 (April 2017), 4–27. DOI:https://doi.org/10.1145/3090064.3090066

[9] Bonchi, Filippo & Bonsangue, Marcello & Rutten, Jan & Silva, Alexandra. (2012). Brzozowski’s Algorithm (Co)Algebraically. 10.1007/978-3-642-29485-3\_2.

[10] J. Sakarovitch. Elements of automata theory. Cambridge University Press, 2009. 6

[11] [DFA minimization - Wikipedia](https://en.wikipedia.org/wiki/DFA_minimization#:~:text=Moore%27s%20algorithm%20Moore%27s%20algorithm%20for%20DFA%20minimization%20is,partition%20until%20no%20more%20refinements%20can%20be%20made.)

[12]

David, J. (2012). Average complexity of Moore's and Hopcroft's algorithms. *Theoretical ComputerScience, 417*, 50-65. Retrieved 12 7, 2021, from https://sciencedirect.com/science/article/pii/s0304397511008814

[13] F. Bassino, J. David, and C. Nicaud. [On the Average Complexity of Moore's State Minimization Algorithm - Inria](https://hal.inria.fr/inria-00359162)

[14] Julien David. [article.dvi (univ-paris13.fr)](https://www-lipn.univ-paris13.fr/~david/articles/mfcs10.pdf)

[15] [Almeida-Zeitoun-Minimization-13xi2006.pdf (cmup.pt)](https://www.cmup.pt/sites/default/files/publications/Almeida-Zeitoun-Minimization-13xi2006.pdf)

[16] T. Kameda and P. Weiner, "On the State Minimization of Nondeterministic Finite Automata," in IEEE Transactions on Computers, vol. C-19, no. 7, pp. 617-627, July 1970, doi: 10.1109/T-C.1970.222994.