2007

State minimization problems in finite state automata

Chris Tauras
West Virginia University

State Minimization Problems in Finite State Automata

by

Chris Tauras

Thesis submitted to the
College of Engineering and Mineral Resources
at West Virginia University
in partial fulfillment of the requirements
for the degree of

Master of Science
in
Computer Science

Lane Department of Computer Science and Electrical Engineering

Morgantown, West Virginia
2007

Keywords: Finite Automata, State Minimization

Abstract

State Minimization Problems in Finite State Automata

by

Chris Tauras
Master of Science in Computer Science

West Virginia University

K. Subramani, Ph.D., Chair

In this thesis, we analyze the problem of state minimization in 2-MDFAs. The class of 2-MDFAs is an extension of the class of DFAs, allowing a small amount of nondeterminism; specifically two start states. Since nondeterminism allows finite automata to be more succinct, it is worthwhile to investigate the problem of minimizing such finite automata. In the case of unbounded non- determinism, i.e., NFAs, such automata can be exponentially more succinct than DFAs [1], but the corresponding minimization problem is PSPACE-complete [2]. Even in the case of 2-MDFAs, which are only polynomially more succinct than DFAs, the minimization problem remains non-trivial; indeed, [3] shows that the corresponding decision problem is NP-complete. We are concerned with the approximability of the 2-MDFA minimization problem. Our main contribution in the current work is the design of an n-factor approximation algorithm for state minimization in 2-MDFAs.

Chapter 1

Introduction

Finite Automata (FAs) are used in many applications such as lexical analysis, parsing, and hashing. As acceptance of a particuliar string can be easily tested in polynomial time, it is desirable to represent the finite automata using the minimum amount of space. A regular language can be represented as a Deterministic Finite Automaton (DFA), a Multiple Start State DFA (MDFA) or as a Non-deterministic Finite Automaton (NFA). While each type of finite automaton represents exactly the class of regular languages, their relative succinctness and the complexities of their minimization problems vary greatly. NFAs can be exponentially more succinct than DFAs [1], but the corresponding minimization problem is PSPACE-complete [2], in contrast to polynomial-time [4]. MDFAs represent an interesting type of finite automata that have recently received much attention in the literature. These FAs have a limited amounted of non-determinism in that there are multiple start states; however, there is precisely one target state for each state on a given input. Even this small amount of nondeterminism in 2-MDFAs is enough to result in a hard minimiztion problem; in particuliar, the problem of state minimization is `NP-complete` for 2-MDFAs. [3]. Unlike general NFAs, which can be exponentially more succinct than DFAs [1], 2-MDFAs are only quadratically more succinct than DFAs (See Section 4). Inasmuch as the state minimization problem in 2-MDFAs is in NP, it is worthwhile to ask whether there exists a polynomial time bounded-error approximation algorithm for this problem. We answer this question in the affirmative, by presenting and analyzing a linear factor approximation algorithm for the same.

The rest of this thesis is organized as follows: Section 2 contains a formal description of the state minimization problem in 2-MDFAs. Section 3 describes the motivation for our work as well as related approaches in the literature. Section 4 describes an n-approximation algorithm for the state minimization problem in 2-MDFAs; as part of the analysis, an algorithm is described for converting a 2-MDFA into a DFA. This section also includes a detailed analysis of the approximation bound. In Section 5, a number of problems are proposed, which are related to the 2-MDFA state minimization problem and provide motivation for the same. Section 6 provides a detailed implementation profile of our algorithm over a wide range of inputs. We conclude in Section 7 by summarizing the main results and detailing problems for future research.

Chapter 2

Statement of Problem

Formally, a 2-MDFA M is defined by the 6-tuple $\langle Q_M, \Sigma, \delta_M, q_s^{0M}, q_s^{1M}, F_M \rangle$, in which:

(a) Q_M denotes the set of states. $|Q_M|$ denotes the *state complexity* of M.

(b) Σ is the input alphabet. Without loss of generality, it is assumed that the alphabet $\Sigma = \{0, 1\}$ for all the finite automata discussed in the remainder of this thesis.

(c) δ_M is the transition function that maps each pair $\langle q, a \rangle \in Q_M \times \Sigma$ to a state $q \in Q_M$, i.e., $\delta : Q_M \times \Sigma \to Q_M$.

(d) q_s^{0M} and q_s^{1M} are the two start states.

(e) F_M is the set of final states.

Figure 2.1 shows an example of a 2-MDFA. It accepts the language $(01)^{*}0 \cup (10)^{*}1 \cup (01)^{*}00(01)^{*}$.

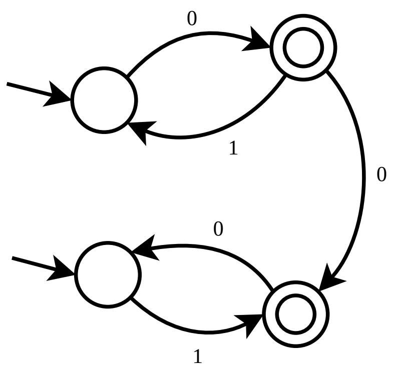

Figure 2.1: An example 2-MDFA $M_0 = M_s$

As with DFAs, we extend δ to $\hat{\delta}$, where $\hat{\delta}$ is a function mapping tuples of states and strings to states. Formally, $\hat{\delta} : Q_M \times \Sigma^* \rightarrow Q_M$ is defined recursively as follows:

(a) $\forall q \in Q_M$, $\hat{\delta}(q, \epsilon) = q$.

(b) $\forall \langle q, a, w\rangle \in Q_M \times \Sigma \ \times \Sigma^*$, $\hat{\delta}(q, wa) = \delta(\hat{\delta}(q, w), a)$.

Definition 2.1 *A string $x \in \Sigma^*$ is said to be* accepted *by the 2-MDFA M, if and only if* $(\{\hat{\delta}(q_s^{0M}, x)\} \cup \{\hat{\delta}(q_s^{1M}, x)\}) \cap F_M \neq \phi$.

Definition 2.2 *The* language *of a 2-MDFA M, denoted by $L(M)$, is the set of all strings $x \in \Sigma^*$ which are accepted by M.*

Given a 2-MDFA $M = \langle Q_M, \Sigma, \delta_M, q_s^{0M}, q_s^{1M}, F_M\rangle$, we define $L(q_s^{0M})$ to be the set of of strings $x \in \Sigma^*$, such that $\{\hat{\delta}_M(q_s^{0M}, x)\} \cap F_M \neq \phi$. In other words, $L(q_s^{0M})$ is the set of strings that are accepted by M, under the provision that the start state is q_s^{0M}. The language $L(q_s^{1M})$ is analogously defined. It is not hard to see that $L(M) = L(q_s^{0M}) \cup L(q_s^{1M})$.

Definition 2.3 *Two 2-MDFAs M_1 and M_2 are* equivalent *if and only if $L(M_1) = L(M_2)$.*

Definition 2.4 *An* ϵ-NFA *is a nondeterministic finite automaton that has the additional ability to transition without consuming the next input character. These spontaneous transitions are known as ϵ-transitions. The addition of this ability does not allow acceptance of any non-regular languages.*

Regardless of the type of automaton M under consideration, we use $L(M)$ to denote the language represented by that automaton and the phrase *state complexity* to denote the number of states Q_M.

Now that all the necessary preliminaries are covered, the problem may now be formally stated:

$\mathbf{P_1}$: *Given a 2-MDFA $M_1 = \langle Q_{M_1}, \Sigma, \delta_{M_1}, q_s^{0M_1}, q_s^{1M_1}, F_{M_1}\rangle$ and a number K, is there a 2-MDFA $M_2 = \langle Q_{M_2}, \Sigma, \delta_{M_2}, q_s^{0M_2}, q_s^{1M_2}, F_{M_2}\rangle$, such that $L(M_1) = L(M_2)$ and $|Q_{M_2}| \leq K$?*

Chapter 3

Motivation and Related Work

The motivation for the study of finite automata comes from many applications, including lexical analysis, hashing, pattern matching, and, of course, the syntax of programming languages [5]. In the case of lexical analysis, compilers for high-level languages, such as Fortran or C, often implement a finite automaton for the lexical analysis for their first pass. Such a finite automaton may possess hundreds or even thousands of states. The large number of states that are potentially used translates to a large amount of memory required by the compiler. Therefore, a method for reducing the number of states would be beneficial in saving space, reducing the cost of implementation. In the case of hashing, the data to be hashed is given as an input to a hashing DFA. The state that the DFA is in, at the end of the string, is the hash value. One way to use a 2-MDFA instead of a DFA for hashing is to use a pair of states after computation, say one state for the high bits and one for the low bits, instead of a single state for the complete value (For instance, see Figure 3.1). The implementation of a 2-MDFA would allow a more complex automaton to be implemented while simultaneously reducing the space requirements. In the case of pattern matching, when strings are concerned, regular expressions are often used to define the desired pattern to match or to search for. On the account of the connection between regular expressions and finite automata, algorithms to do this often make use of finite automata to perform the search or match operation, such as in [6] and [7]. It should be noted that automata with more states require more space to implement, so it is desirable to reduce the number of states needed.

The problem of state minimization has been studied extensively in a number of different types of finite automata. It is well-known that the problem of minimizing DFAs is solvable in polynomial time [4], whereas the problem of minimizing NFAs is `PSPACE-complete` [2].

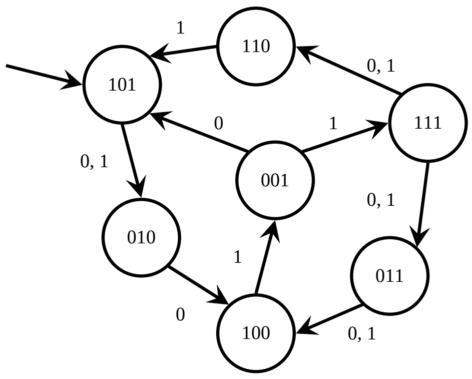

Figure 3.1: Using finite automata in hashing

In an effort to determine how much nondeterminism can be permitted in finite automata while keeping the state minimization problem easy, several classes of finite automata with limited nondeterminism have been studied. Unambiguous Finite State Automata (UFAs) form one such class; for any input string, there is never more than one accepting path. The problem of minimizing the number of states in a UFA has been shown to be `NP-complete` [1], as well as converting a DFA to a minimal UFA [8]. In [3], the complexity of state minimization problems in k-MDFAs was investigated and shown to be `NP-complete` for $k \geq 2$, where k denotes the number of distinct start states in the MDFA. We focus on the problem on minimizing the number of states in a 2-MDFA, with a view towards obtaining non-trivial approximation bounds; additionally, the the degree of succinctness of 2-MDFAs is explored. The complexity inherent in representing regular languages succinctly has been studied in [9, 10] and [11]. [12] relates the type of ambiguity of finite automata to the succinctness of their representation. We thus see that determining succinct (minimal) representations of languages is of enormous interest to the Automata Theory community.

The difficulty of dealing with `NP-Hard` optimization problems can be mitigated to some extent by devising efficient approximation algorithms [13]. Basically, a polynomial time approximation algorithm for an optimization (minimization) problem delivers an output in polynomial time, but with a certain loss of accuracy. Such algorithms are useful in applications in which accuracy can be sacrificed to some extent, if the output is computed quickly. The relevance of these algorithms and their practical impact are discussed at length in [13].

Definition 3.1 *Let Π denote a minimization problem, and let OPT denote an optimal algorithm for the same. For an arbitrary instance $I \in \Pi$, $OPT(I)$ refers to the output of OPT. Let*

$\mathcal{A}$ denote an algorithm for Π, with $\mathcal{A}(I)$ denoting the output of algorithm $\mathcal{A}$ on an arbitrary instance $I \in \Pi$. $\mathcal{A}$ is said to be a polynomial time approximation algorithm for Π, if:

(a) $\mathcal{A}$ runs in polynomial time.

(b) For all instances $I \in \Pi$, $\mathcal{A}(I) \leq c \cdot OPT(I)$, where c is called the factor of approximation.

Provided that the above conditions are met, algorithm $\mathcal{A}$ is said to be a c*-factor approximation algorithm* for the problem Π.

The analysis of an approximation algorithm typically proceeds in two steps. First it is shown that for all instances $I \in \Pi$, $\mathcal{A}(I) \leq c_1$; then it is shown that for all instances $I \in \Pi$, $OPT(I) \geq c_2$. From these two proofs, we can conclude that for all instances $I \in \Pi$, $\mathcal{A}(I) \leq \frac{c_1}{c_2} \cdot OPT(I)$. *It is important to note that our analysis is significantly different from the traditional analyses of approximation algorithms.*

Within the field of Automata theory, there has been no active effort to study approximation algorithms as a tool; indeed we could find only one result [14], and that result was concerned with the *inapproximability* of the Minimum Consistent Finite Automaton problem [15].

Our work establishes that there exists an n-factor approximation algorithm for problem $\mathbf{P_1}$, where n denotes the minimum number of states required to represent the input language as a 2-MDFA.

Chapter 4

Algorithm and Analysis

We shall now present our algorithm for approximating the minimum number of states required to represent the input language as a 2-MDFA. At the heart of our algorithm is a procedure that converts a 2-MDFA into a DFA. We shall argue that the conversion results in only a quadratic increase in the number of states; this is in contrast to the procedure which converts an NFA into a DFA, in which the state blowup could be exponential.

Our algorithm proceeds as follows: First, the input 2-MDFA M_0 is pruned; the pruning procedure consists of eliminating redundant and unreachable states. The details of the procedure are described in SHRINK-2MDFA$()$ (Algorithm 4.3). The resultant 2-MDFA M_s is then converted into an ϵ-NFA M_1'. Subsequently, M_1' is converted into an NFA M_2 and then into a DFA M_3. This is followed by M_3 being minimized to obtain M_4. Finally, we return either M_s or M_4, depending on which of the two has smaller state complexity.

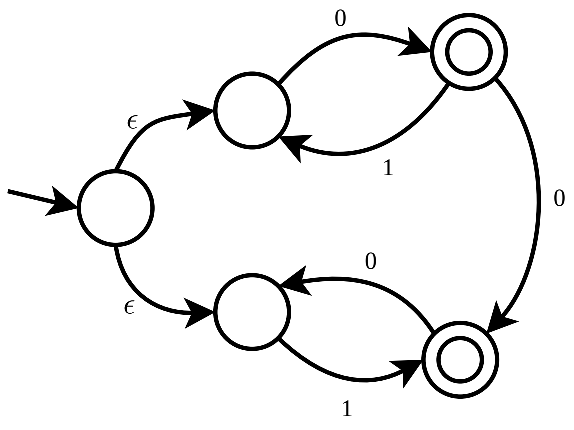

Figure 4.1: M_1'

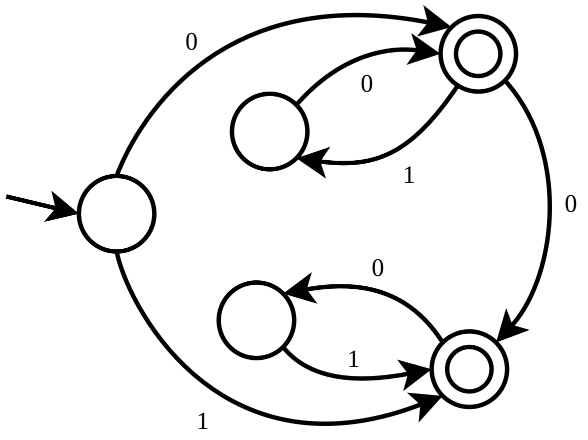

Figure 4.2: $M_2 = M_3 = M_4$

```
Function MINIMIZE ($M_0$ = $\langle Q_{M_0}, \Sigma, \delta_{M_0}, q_s^{0M_0}, q_s^{1M_0}, F_{M_0}\rangle$)
 1: Let $M_s$ ($\langle Q_{M_s}, \Sigma, \delta_{M_s}, q_s^{0M_s}, q_s^{1M_s, F_{M_s}}\rangle$) $=$ SHRINK-2MDFA($M_0$).
 2: Let $M_1'$ ($\langle Q_{M_1'}, \Sigma, \delta_{M_1'}, q_s^{M_1'}, F_{M_1'}\rangle$) $=$ MAKE-$\epsilon$-NFA($M_s$)
 3: Compute the $\epsilon$-closure of $q_s^{M_1'}$ to convert $M_1'$ into an NFA $M_2$ ($\langle Q_{M_2}, \Sigma, \delta_{M_2}, q_s^{M_2}, F_{M_2}\rangle$).
 4: Convert $M_2$ into a DFA $M_3$ ($\langle Q_{M_3}, \Sigma, \delta_{M_3}, q_s^{M_3}, F_{M_3}\rangle$).
 5: Minimize $M_3$ to get $M_4$ ($\langle Q_{M_4}, \Sigma, \delta_{M_4}, q_s^{M_4}, F_{M_4}\rangle$), using the DFA minimization algorithm
    in [16].
 6: if ($|Q_{M_4}| \leq |Q_{M_s}|$) then
 7:     return $M_{out}$ ($\langle Q_{M_{out}}, \Sigma, \delta_{M_{out}}, q_s^{M_{out}}, F_{M_{out}}\rangle$) $= M_4$.
 8: else
 9:     return $M_{out}$ ($\langle Q_{M_{out}}, \Sigma, \delta_{M_{out}}, q_s^{0M_{out}}, q_s^{1M_{out}}, F_{M_{out}}\rangle$) $= M_s$.
10: end if
```

Algorithm 4.1: 2-MDFA Minimization

```
Function MAKE-$\epsilon$-NFA ($M_s$ = $\langle Q_{M_s}, \Sigma, \delta_{M_s}, q_s^{0M_s}, q_s^{1M_s}, F_{M_s}\rangle$)
 1: Insert a new start state $q_s^{M_s}$ into $M_s$.
 2: Insert $\epsilon$-transitions from $q_s^{M_s}$ to $q_s^{0M_s}$ and $q_s^{1M_s}$.
 3: $\{q_s^{0M_s}$ and $q_s^{1M_s}$ are no longer start states.$\}$
 4: return $M_s$
```

Algorithm 4.2: Conversion to ϵ-NFA

The purpose of the SHRINK-2MDFA() function is to eliminate useless states from the input 2-MDFA.

Definition 4.1 *Two states p and q are said to be equivalent in a finite automaton M, if, for all strings $w \in \Sigma^*$, $\hat{\delta}_M(p, w)$ leads to an accepting state in M if and only if $\hat{\delta}_M(q, w)$ does.*

Any pair of equivalent states p and q may be merged without affecting the language of the finite automaton M in question. This is because the remainder of the input string w will either be accepted by both states or be rejected by both states.

Definition 4.2 *A state p in a 2-MDFA M_0 is said to be* reachable *if there is a directed path from either $q_s^{0M_0}$ or $q_s^{1M_0}$ to p in the directed graph representing M_0.*

A state which is not reachable from either start state is called *unreachable*. Clearly, such states cannot possibly be part of the computational path for any input string, so the language of the 2-MDFA in question will not change if such states are removed.

```
Function SHRINK-2MDFA ($M_0$ = $\langle Q_{M_0}, \Sigma, \delta_{M_0}, q_s^{0M_0}, q_s^{1M_0}, F_{M_0}\rangle$)
1: Let $M_1$ denote a copy of $M_0$
2: Replace each block of equivalent states with a single state, using the algorithm in [16].
3: Delete all unreachable states from $M_1$.
4: return $M_1$.
```

Algorithm 4.3: Shrinking a 2-MDFA

For instance, let the 2-MDFA represented by Figure 2.1 be the input M_0 to Algorithm 4.2. Since M_0 has no pairs of equivalent states and no unreachable states, the pruning procedure does not alter it and $M_s = M_0$. Figure 4.1 displays the 2-MDFA after it has been converted to an ϵ-NFA M_1'. M_1' is then converted to the NFA M_2 as shown in Figure 4.2. It so happens that M_2 is already in minimized DFA form, and, therefore, no further work needs to be done. Finally, M_s has fewer states than M_4 and, hence, M_s is returned by Algorithm 4.2.

Observation 4.1 *Consider the NFA M_2 = $\langle Q_{M_2}, \Sigma, \delta_{M_2}, q_s^{M_2} F_{M_2}\rangle$ in Line 4 of Algorithm 4.1. Observe that converting the ϵ-NFA M_1' to an NFA M_2 involves the following steps: Compute the ϵ-closure S of $q_s^{M_1'}$; observe that S contains $q_s^{M_1'}$ and the two states, say q_r and q_t, to which M_1' can move from $q_s^{M_1'}$ on ϵ-transitions. Thus, it is clear that $|S| = 3$. $q_s^{M_1'}$ is replaced with a new start state $q_s^{M_2}$. For each symbol $a \in \Sigma$, we set $\delta(q_s^{M_2}, a) = \delta(q_r^{M_1'}, a) \cup \delta(q_t^{M_1'}, a)$.*

The remaining states and transitions are identical in M_1' and M_2. Note that this construction exploits the special structure of M_1'. For any symbol $a \in \Sigma$, we observe that $|\delta_{M_2}(q_s^{M_2}, a)| \leq 2$, whereas, for all other states $q \in Q_{M_2}$, $|\delta_{M_2}(q, a)| \leq 1$. In other words, the non-determinism of M_2 is limited to the first move.

Observation 4.2 *There does not exist a directed path from any state in the ϵ-NFA M_1' to the start state $q_s^{M_1'}$. Therefore, it follows that the start state $q_s^{M_2}$ of M_2 is not reachable from any other state in M_2.*

Lemma 4.1 *For any string $x \in \Sigma^*$, $|\hat{\delta}_{M_2}(q_s^{M_2}, x)| \leq 2$.*

Proof. We prove Lemma 4.1 by using induction on the length of the input string x.

BASE CASE: $|x| = 0$ and, hence, $x = \epsilon$. Since $x = \epsilon$, it follows that M_2 does not make any move and stays in the start state. In other words, $|\hat{\delta}_{M_2}(q_s^{M_2}, x)| = 1 \leq 2$.

INDUCTIVE STEP: Assume that $|\hat{\delta}_{M_2}(q_s^{M_2}, y)| \leq 2$ whenever M_2 is presented with a string y such that $|y| \leq n - 1$. Now consider a string, $x = wa$, with $|x| = n$, $|w| = n - 1$ and $a \in \Sigma$. Let us define $Q_\delta = \hat{\delta}_{M_2}(q_s^{M_2}, w)$. As per the inductive hypothesis, $|Q_\delta| \leq 2$. Without loss of generality, let us assume that Q_δ consists of exactly two states, say q_u and q_v. Note that transitions out of q_u and q_v are deterministic, i.e. $|\hat{\delta}_{M_2}(q_u, a)| = 1$ and $|\hat{\delta}_{M_2}(q_v, a)| = 1$. Therefore, $|\hat{\delta}_{M_2}(q_s^{M_2}, x)| = |Q_\delta| \leq 2$, and the claim follows. □

Lemma 4.2 *The number of states in M_3 is $O(|Q_{M_s}|^2)$.*

Proof. Note that the general algorithm for converting an NFA to a DFA enumerates all possible subsets of states in which the NFA can exist. In general, there is an exponential number of subsets to enumerate. However, in this case, the NFA M_2 can exist in at most two states after reading an input string; consequently, the conversion algorithm need only enumerate those subsets that contain no more than two states. Hence, the number of states that need to be enumerated is $O(|Q_s|^2)$, which also represents the state complexity of M_3. □

Theorem 4.1 *Algorithm 4.1 runs in time polynomial in the size of its input.*

Proof. Each step of Function MINIMIZE() can be implemented to run in time that is polynomial in the size of the input. It is resonable to consider the number of states of the input 2-MDFA the “size” n of the input. Although the representation of each state technically needs $O(\log n)$

bits, and although the transition function needs to be included in the input, these things would only result in an input size of $O(n^2 \log n)$ bits, so, for simplicity, these things may be ignored.

Line 1 involves eliminating blocks of equivalent states and can be implemented in $O(n^3)$ time, as per the algorithm in [16]. The conversion of M_s into an ϵ-NFA M_1' is a constant time operation with the addition of one state; likewise, as discussed above, the conversion of M_1' into an NFA M_2 is a constant time operation. Line 4 involves the conversion of the NFA M_2 into a DFA M_3; since we have to enumerate state pairs only, this operation can be implemented in $O(n^3)$ time. Note that M_3 has $O(n^2)$ states; it follows that the DFA minimization procedure in Line 5 can be implemented in time $O(n^6)$ time [16].

Thus, Algorithm 4.1 can be implemented in time that is polynomial in the size of its input. □

Theorem 4.2 *Let $L(M_0)$ denote the regular language represented by the input 2-MDFA M_0 and let $n = |Q_{M_4}|$ denote the number of states in the minimized 2-MDFA M_4 (See Algorithm 4.1). The optimal 2-MDFA for M_1, i.e., the 2-MDFA with the fewest number of states representing $L(M_0)$, must have $\Omega(n^{\frac{1}{2}})$ states.*

Proof. Let S_{EM_0} denote the set of all 2-MDFAs that are equivalent to the input 2-MDFA M_0. The minimum state DFA corresponding to a given language $L(M_0)$ is unique [16]; accordingly, regardless of the 2-MDFA in S_{EM_0} that is presented as input to Algorithm 4.1, the number of states in the minimized DFA M_4 is the same. Let $M_{opt} = \langle Q_{M_{opt}}, \Sigma, \delta_{M_{opt}}, q_s^{0M_{opt}}, q_s^{1M_{opt}}, F_{M_{opt}} \rangle$ denote the optimal 2-MDFA corresponding to $L(M_0)$ Clearly, $M_{opt} \in S_{EM_0}$; further, $|Q_{opt}| \leq |Q_{M_r}|,\ \forall M_r \in S_{EM_0}$. Let us focus on the situation in which M_{opt} is presented to Algorithm 4.1. The corresponding M_4 would be no larger than $O(|Q_{M_{opt}}|^2)$ as per the discussion above. But M_4 and, therefore, $|Q_{M_4}|$ are the same for all 2-MDFAs $M_r \in S_{EM_0}$; note that this includes M_0. Therefore, it follows that the output of Algorithm 4.1 on input M_0 has at most $O(|Q_{M_{opt}}|^2)$ states. In other words, if the output of Algorithm 4.1 on an input has n states, then the optimal 2-MDFA for this instance has $\Omega(n^{\frac{1}{2}})$ states. □

Corollary 4.1 *Algorithm 4.1 is a linear factor approximation algorithm, where the linear factor refers to the size of the optimal 2-MDFA.*

Proof. From Theorem 4.2 and the discussion in Section 3, it is clear that the output produced by Algorithm 4.1 is off by at most a linear factor from the optimum; in other words, if the state complexity of M_{opt} is p, then the state complexity of M_{opt}, i.e., the automaton that is output by Algorithm 4.1, is at most p^2. The claim follows. $\Box$

The following theorem will show that the linear factor bound is tight for Algorithm 4.1; thus any improvement in the approximation bound will require the development of new techniques.

Theorem 4.3 *There exists a* 2*-MDFA* M_0 *such that the optimal* 2*-MDFA for* $L(M_0)$ *has state complexity* p*, while the output of Algorithm 4.1 on* M_0 *has state complexity* p^2*.*

Proof. Let $M_{d1} = \langle Q_{M_{d1}}, \Sigma, \delta_{M_{d1}}, q_s^{M_{d1}}, F_{M_{d1}} \rangle$ and $M_{d2} = \langle Q_{M_{d2}}, \Sigma, \delta_{M_{d2}}, q_s^{M_{d2}}, F_{M_{d2}} \rangle$ denote two minimal DFAs representing the languages $L(M_{d1})$ and $L(M_{d2})$ respectively, such that:

(a) The minimal DFA $M_{d3} = \langle Q_{M_{d3}}, \Sigma, \delta_{M_{d3}}, q_s^{M_{d3}}, F_{M_{d3}} \rangle$ representing the language $L(M_{d1}) \cup L(M_{d2})$ has state complexity $\Theta(|Q_{M_{d1}}| \cdot |Q_{M_{d2}}|)$.

(b) $|Q_{M_{d1}}| = \Theta(|Q_{M_{d2}}|)$.

It is well-known that such languages exist; for instance, see [9], where the properties of such languages are discussed. The minimal 2-MDFA for the language $L(M_{d3})$ clearly has state complexity $O(|Q_{M_{d1}}|)$. However, we can construct a 2-MDFA $M_0 = \langle Q_{M_0}, \Sigma, \delta_{M_0}, q_s^{0M_0}, q_s^{1M_0}, F_{M_0} \rangle$ with the following properties:

(i) $L(q_s^{0M_0}) = L(M_{d3})$.

(ii) $L(q_s^{1M_0}) = \epsilon$.

Note that $L(M_0) = L(q_s^{0M_0}) \cup L(q_s^{0M_0}) = L(M_{d3})$. Secondly, the DFA with start state $q_s^{0M_0}$ has a transition function which is identical to the transition function of the DFA M_{d3}, i.e., the state complexity of M_0 is $\Theta(|Q_{M_{d1}}| \cdot |Q_{M_{d2}}|) = \Theta(|Q_{M_{d1}}|^2)$.

Accordingly, the SHRINK-2MDFA() procedure leaves M_0 unaltered, and the DFA minimization procedure returns a DFA with $\Theta(|Q_{M_{d1}}|^2|)$ states. Since both procedures return a 2-MDFA with $\Theta(|Q_{M_{d1}}|^2|)$ states, Algorithm 4.1 necessarily returns a 2-MDFA with $\Theta(|Q_{M_{d1}}|^2|)$ states. However, the optimal 2-MDFA for the language $L(M_{d3})$ has $O(|Q_{M_{d1}}|)$ states, and the theorem follows. $\Box$

The next theorem shows that Algorithm 4.1 is optimal in certain cases.

Theorem 4.4 *There exists a regular language L_0, such that Algorithm 4.1 computes the optimal*
2-MDFA when presented with any input 2-MDFA that accepts L_0.

Proof. Let $L_0 = \Sigma^*$. The optimal DFA for L_0 has precisely one state. Since this DFA is unique, the state complexity of the automaton returned by Algorithm 4.1 is always 1, regardless of the state complexity of the 2-MDFA that is used to represent L_0. □

We make the following observations about Algorithm 4.1:

(1) The state complexity of the output of Algorithm 4.1 (M_{out}) is never greater than the state complexity of its input (M_0). Note that the SHRINK-2MDFA() procedure does not increase state complexity and, hence, $|Q_s| \leq |Q_0|$. Algorithm 4.1 returns either M_s or the minimized DFA M_4 depending on which automaton has smaller state complexity.

(2) It follows that Algorithm 4.1 is optimal when the minimal 2-MDFA is given as the input.

From the empirical perspective (see Section 6), it appears that for a given 2-MDFA M_0, the SHRINK-2MDFA() procedure returns a better quality of approximation than the DFA minimization procedure. This suggests that by merely computing M_s, a good approximation is generally obtained; however, the following lemma proves that this is not true in general.

Lemma 4.3 *As an approximation to the optimal 2-MDFA, the state complexity of the output of the* SHRINK-2MDFA() *procedure is arbitrarily bad.*

Proof. Let M_0 denote a 2-MDFA, with $L(q_s^{0M_0})$ representing an arbitrary language with an arbitrary DFA state complexity m. Let $L(q_s^{1M_0}) = \Sigma^*$. Note that the minimal 2-MDFA for M_0 has only one state that transitions to itself on all strings $x \in \Sigma^*$. When M_s is computed from M_0 using the SHRINK-2MDFA() procedure, the language of the 2-MDFA start states is not redefined, and, hence, $L(q_s^{0M_0})$ and $L(q_s^{1M_0})$ remain the same. Therefore, M_s has state complexity $\Theta(m)$, which is arbitrarily bad as an approximation. □

The principal drawback of the SHRINK-2MDFA() procedure is that it does not alter the languages of the individual start states.

Chapter 5

Related Problems

In this section, we discuss a number of problems related to 2-MDFA minimization. While pertinent to the main problem discussed in this thesis, these state optimization problems are also interesting in their own right.

We first consider the problem of reducing the states in a DFA by converting it into a 2-MDFA and establish its complexity.

Lemma 5.1 *There exists no polynomial time algorithm for the problem of converting a DFA M_d to a minimal 2-MDFA M_t, unless* P = NP.

Proof. Assume that there exists an algorithm $\mathcal{A}$, that takes as input an arbitrary DFA M_0 and returns the optimal state state 2-MDFA, representing $L(M_0)$ in polynomial time. We can use $\mathcal{A}$ to obtain a polynomial time algorithm for the 2-MDFA state minimization problem as follows: Given an arbitrary 2-MDFA M_1, convert it into a DFA M_{d1} in polynomial time (as discussed in Section 4), and then provide it as input to Algorithm $\mathcal{A}$ to compute the optimal state 2-MDFA for the same language. However, this would mean that P = NP, since the 2-MDFA minimization problem is NP-complete. □

We now consider the Optimal Splitting and Optimal Merging problems.

(a) Optimal Splitting: Assume that we are given a DFA M_0, which is the minimal DFA for the language $L(M_0)$ and a number K. The *Optimal Splitting Problem* is concerned with splitting $L(M_0)$ into two languages $L(M_1)$ and $L(M_2)$, represented by minimal DFAs M_1 and M_2 respectively, such that $|Q_{M_1} + |Q_{M_2}| \leq K$. This problem is neither known to be `NP-complete` nor known to have a constant factor approximation. The optimal split number $split_{M_0}$ for a given minimal DFA M_0 is the smallest value of K such that $L(M_0)$ can be split into two distinct languages, L_1 and L_2.

(b) Optimal Pairing: Assume that we are given two DFAs, M_1 and M_2, representing the languages $L(M_1)$ and $L(M_2)$, respectively, and a number K. In the *Optimal Pairing Problem*, the goal is to find two new DFA's M'_1 and M'_2 such that:

(a) $L(M_1) = L(M'_1)$.

(b) $L(M_2) = L(M'_2)$.

(c) When equivalent states are merged across M'_1 and M'_2, the total number of states in the resultant 2-MDFA is at most K.

It is important to note that minimizing M_1 and M_2 does not necessarily provide the optimal solution. This is because the sub-optimal DFAs of two languages may have more states in common than their corresponding minimal versions.

Chapter 6

Implementation Results

In this section, we discuss our empirical observations on the effectiveness of Algorithm 4.1 from the perspective of 2-MDFA minimization.

6.1 Experimental Setup

2-MDFAs were represented as graph data structures; for instance, see [17]. For the sake of uniformity in the comparions, all automata had exactly 200 states in our experiments. There were two probabilities associated with each 2-MDFA instance:

(i) The *finality probability*, P_f - Attached to each state of the automaton, this measure represents the probability that the state is a final state.

(ii) The *transition probability*, P_t - This measure represents the probability that a given transition from a state exists. For instance, if it is determined that there exists a transition on input 0 from a given state, then the said transition is equally likely to move the automaton from the given state to any state other than the dead state. If it is determined that no transition exists from a given state on a given input, then a transition to the dead state is inserted.

In our experiments, we generated three types of 2-MDFA instances: dense automata, sparse automata, and intermediate automata. Dense automata were generated with $P_t = 1$ and $P_f = \frac{1}{2}$; sparse automata were generated with $P_t = \frac{1}{2}$ and $P_f = \frac{1}{2}$; and, finally, intermediate automata were generated with $P_t = \frac{4}{5}$ and $P_f = \frac{1}{2}$. Random instances of 2-MDFAs were generated using

$\lvert Q_{M_0}\rvert$ (input)	$\lvert Q_{M_s}\rvert$	$\lvert Q_{M_4}\rvert$	$\lvert Q_{M_{out}}\rvert$ (output)
200	154	7533	154
200	170	10178	170
200	161	9152	161
200	164	9396	164
200	152	7670	152
200	148	7866	148
200	168	9895	168
200	163	8440	163
200	158	8613	158
200	159	7090	159

Table 6.1: Implementation profile over dense automata

both a linear congruential generator (LCG) and an inversive congruential generator (ICG). For each class of 2-MDFAs, the first five results were obtained with the LCG, and the remaining five were obtained using the ICG. The lack of discrepancy between the results from the two generators allows us to gain confidence that these generators are indeed a good approximation of true randomness, at least as far as these results are concerned.

6.2 Observations on Dense Automata

For our first test, we generated random instances of dense automata; i.e. automata that have a high expected number of transitions. In each instance, the pruning procedure was much more effective than the DFA conversion; in particular, the pruning procedure produced automata with at most 170 states, whereas the DFA minimization can easily produce automata with over 10000 states (see Table 6.1). As opposed to the standard representation of a regular language (a minimal DFA), Algorithm 4.1 produces a much smaller output.

$\|Q_{M_0}\|$ (input)	$\|Q_{M_s}\|$	$\|Q_{M_4}\|$	$\|Q_{M_{out}}\|$ (output)
200	2	2	2
200	25	24	24
200	15	15	15
200	2	2	2
200	2	2	2
200	5	5	5
200	16	16	16
200	40	41	40
200	15	15	15
200	4	3	3

Table 6.2: Implementation profile over sparse automata

6.3 Observations on Sparse Automata

For our second test, we generated random instances of sparse automata; i.e. automata that have a very low expected number of transitions. In this case, the pruning procedure and the DFA conversion procedure were almost exactly equal; neither was significantly better than the other (see Table 6.2). Although, in this case Algorithm 4.1 is no better than the standard representation of a regular language (a minimal DFA), it is at least no worse.

6.4 Observations on Intermediate Automata

For our third test, we generated random instances of intermediate automata; in this case, the expected number of transitions was between that of the dense and sparse automata. The intent of this was to generate some cases in which the pruning procedure is somewhat better but not strikingly better than the DFA conversion procedure. Instead, a more interesting result was obtained: either the pruning procedure was much better then the DFA conversion procedure or it was not significantly better than the DFA minimization procedure (see Table 6.3). It is curious that the anticipated result of the pruning procedure being somewhat superior was not obtained in any run. However, the pruning procedure was more effective overall, and Algorithm 4.1,

$\lvert Q_{M_0}\rvert$ (input)	$\lvert Q_{M_s}\rvert$	$\lvert Q_{M_4}\rvert$	$\lvert Q_{M_{out}}\rvert$ (output)
200	129	2806	129
200	136	137	136
200	97	1132	97
200	104	1692	104
200	122	123	122
200	117	2378	117
200	131	137	131
200	114	2099	114
200	149	6002	149
200	109	1690	109

Table 6.3: Implementation profile over intermediate automata

in many cases, produces a much smaller output than the standard representation of a regular languge (a minimal DFA).

Chapter 7

Conclusions

Our main result is an approximation algorithm to the problem of state minimization for an arbitrary 2-MDFA. This algorithm is bounded by a linear factor and is guaranteed to have an output no larger than any equivalent DFA. This is the first non-trivial, *positive* result that we are aware of, insofar as approximation algorithms for NP-Hard optimization problems in Automata Theory are concerned. As part of our analysis, we showed that 2-MDFAs are only polynomially as succinct as DFAs, with respect to representing a given regular language. We established that our approximation algorithm is tight; we have shown there exist regular languages for which the state complexity of the output automaton is larger by no less than a linear factor from the state complexity of the optimal 2-MDFA. We also proposed some interesting problems with unknown computational complexity which are related to 2-MDFA minimization.

Hopefully, this result will stimulate more interest in approximation algorithms for hard minimization problems in automata theory. Towards this end, we propose two open problems that are good candidates for a further extension of our results.

(a) Is there a fast, constant factor approximation algorithm for $\mathbf{P_1}$? - The existence of such an algorithm has enormous practical significance. An alternative line of research is to establish that such an algorithm cannot exist unless P = NP.

(b) How much savings does our algorithm provide on practical instances? - The implementation profile in Section 6 was derived using random instances; an empirical study over practical instances would serve as a dependable baseline for future empirical studies.

References

[1] A. Meyer and M. Fischer, "Economies of description by automata, grammars, and formal systems," in *Proceedings of the 12th SWAT (Annual Symposium on Switching and Automata Theory)*, 1971, pp. 188–191.

[2] T. Jiang and B. Ravikumar, "Minimal NFA problems are hard," in *Proceedings of the 18th International Colloquium on Automata, Languages and Programming, ICALP'91 (Madrid, Spain, July 8-12, 1991)*, ser. LNCS, J. L. Albert and M. R. Artalejo, Eds. Berlin-Göttingen-Heidelberg-New York: Springer-Verlag, 1991, vol. 510, pp. 629–640.

[3] A. Malcher, "Minimizing finite automata is computationally hard," *Theoretical Computer Science*, vol. 327, no. 3, pp. 375–390, Nov. 2004.

[4] J. E. Hopcroft, "An $n \log n$ algorithm for minimizing the states in a finite-automaton," in *Theory of Machines and Computations*, Z. Kohavi, Ed. Academic Press, 1971, pp. 189–196.

[5] K. C. Louden, *Programming Languages: Principles and Practice*. Brooks/Cole, 2002.

[6] C. L. A. Clarke and G. V. Cormack, "On the use of regular expressions for searching text," *ACM Trans. Program. Lang. Syst.*, vol. 19, no. 3, pp. 413–426, 1997.

[7] R. A. Baeza-Yates and G. H. Gonnet, "Fast text searching for regular expressions or automaton searching on tries," *J. ACM*, vol. 43, no. 6, pp. 915–936, 1996.

[8] T. Jiang and B. Ravikumar, "Minimal nfa problems are hard," *SIAM J. Comput.*, vol. 22, no. 6, pp. 1117–1141, 1993.

[9] S. Yu, Q. Zhuang, and K. Salomaa, "The state complexities of some basic operations on regular languages." *Theor. Comput. Sci.*, vol. 125, no. 2, pp. 315–328, 1994.

[10] E. L. Leiss, "Succint representation of regular languages by boolean automata." *Theor. Comput. Sci.*, vol. 13, pp. 323–330, 1981.

[11] Meyer and Fischer, "Economy of description by automata, grammars, and formal systems," in *FOCS: IEEE Symposium on Foundations of Computer Science (FOCS)*, 1971.

[12] B. Ravikumar and O. H. Ibarra, "Relating the type of ambiguity of finite automata to the succinctness of their representation." *SIAM J. Comput.*, vol. 18, no. 6, pp. 1263–1282, 1989.

[13] D. Hochbaum, Ed., *Approximation Algorithms for NP-Hard Problems*. Boston, Masachusetts: PWS Publishing Company, 1996.

[14] H. U. Simon, "On approximate solution for combinatorial optimization problems," *SIAM Journal of Discrete Mathematics*, vol. 3, pp. 294–310, 1990.

[15] P. Crescenzi and V. Kann, "Approximation compendium," http://www.nada.kth.se/˜viggo/wwwcompendium/node242.html.

[16] J. E. Hopcroft, R. Motwani, and J. D. Ullman, *"Introduction to Automata Theory, Language, and Computation"*, 2nd ed. Addison–Wesley, 2001.

[17] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms*, 2nd ed. Boston, Massachusetts: MIT Press and McGraw-Hill Book Company, 1992.