International Journal of Modern Trends in Engineering and Research
www.ijmter.com
e-ISSN No.:2349-9745, Date: 2-4 July, 2015

A Fast and Memory Efficient Pattern Matching Using Deterministic Finite Automata Compression Approach

Miss. Utkarsha P. Pisolkar[1], Asst. Prof. Shivaji R. Lahane[2]

[1]*Computer Engg. Department, GES's R. H. Sapat College of Engineering, Management Studies and Research Nashik, Affiliated to Savitribai Phule Pune University, Maharashtra, India, utkarshappisolkar@gmail.com*
[2]*Computer Engg. Department, GES's R. H. Sapat College of Engineering, Management Studies and Research, Nashik, Affiliated to Savitribai Phule Pune University, Maharashtra, India, shivajilahane@gmail.com*

Abstract- Patten matching algorithm often uses Deterministic finite automaton to represent interested patterns. Regular expression representation of patterns is expressive and compact. Memory requirement of Deterministic finite automata is important factor in pattern matching algorithms. The method described in this paper reduces size of Aho-Corasick Deterministic finite automata which is generated from regular expressions and uses this compressed Deterministic finite automata in pattern matching process to speed up it. The compression approach uses bit reduction method which decreases the rules to only one rule for each state by representing all transitions to that state through single prefix.

Keywords- pattern matching; deterministic finite automata; regular expression; aho-corasick; bit reduction.

I. INTRODUCTION

Pattern matching algorithms are very widely used in networking and security areas. Deterministic finite automaton (DFA) is often used to represent patterns in pattern matching algorithms. DFA is traversed during pattern matching process to check existence of pattern. Therefore, size of DFA is always an important issue in pattern matching algorithm which affects on speed of pattern matching process [1]. The Aho-Corasick is a very widely used algorithm for DFA construction and pattern matching [2]. The DFA is used in Aho-Corasick algorithm to represent interested patterns. At the time of pattern matching, the input text is verified symbol by symbol by traversing the DFA. The direct implementation of DFA stores a rule for each cases that is, one rule per every pair of a state and a symbol. This involves vast memory requirement [1]. Therefore it is important to reduce size of DFA so that it won't result in vast memory requirement and more pattern matching time. This paper describes an approach for compression of DFA generated from regular expression to achieve memory efficient and faster pattern matching process. The compression approach decreases the rules of each state by representing all transitions to a particular state by a single prefix that captures a set of current states [1]. This approach has three main steps as DFA construction, DFA compression and Pattern matching. The DFA compression step compresses Aho-Corasick DFA using CompactDFA algorithm and produces compressed rules to store DFA. These compressed rules are later on used in pattern matching process. The remaining paper is organized as follows. Survey of literature of some of the existing compression techniques and various algorithms used till today are discussed in Section two. Section Three highlights the DFA compression approach in detail, and Section four presents partial results of DFA compression system on dataset. Concluding remarks and future work appear in Section five.

II. LITERATURE SURVEY

DFA compressions approaches used till today are mainly contain techniques like decreasing number of transitions, number of states, alphabet sets and bits encoding the transitions. The D^2FA [3] and CD^2FA [4] approaches have decreased number of transitions of state. The D^2FA [3] is the Delayed Input DFA (D^2FA) technique, which reduced space requirements by reducing the number of distinct transitions between states by single default transition [3]. The CD^2FA [4] is used to boost the speed of D^2FA as by storing more information on the transitions edges. In Content Addressed Delayed Input DFA (CD^2FA), state numbers are replaced by content label to skip past default transitions.

The Hybrid DFA-NFA, HFA and XFA approaches have decreased number of states. The hybrid DFA-NFA [5] is state reduction solution. A hybrid DFA-NFA solution was combined the advantages of both automata DFA and NFA [5]. NFA was used for state explosion nodes and DFA was used for remaining nodes. A History-based Finite Automaton (H-FA) has stored the transition history and there by decreased the number of states. It has stored the history information in a history buffer which is a small and fast cache [6]. The other state reduction technique is extended character set (XFA) [7]. To eliminate conditional transitions it has used many automata transformations. XFA algorithm consists of one state per regular expression, which is very complex for difficult regular expressions.

The Delta finite automaton [8] is an alternative representation for DFA. The algorithm has trim down the number of states as well as transitions. This algorithm has stored only differences between most adjacent states which shares many common transitions [8]. The alphabet compression table approach [9] has converted the set of characters in an input alphabet set to a minor set of clustered characters that has the similar transitions for some states in the automaton. This approach has created a discrete alphabet compression table for every partition [9]. HEXA [10] is the History based Encoding, eXecution and Addressing (HEXA) approach. It has reduced the number of bits that represents every state and has used implicit information to reduce the information that must be stored explicitly [10]. In DFA, multiple paths leading to each node are stored in history [10]. This HEXA approach was applied on data structures used by Aho-Corasick algorithm and bit-split side of algorithm [11].

III. DFA COMPRESSION APPROACH

The DFA compression approach reduces the size of DFA generated from regular expression. The compression approach consists of three main steps: DFA construction from snort dataset [12], DFA compression and Pattern matching as shown in Figure 1.

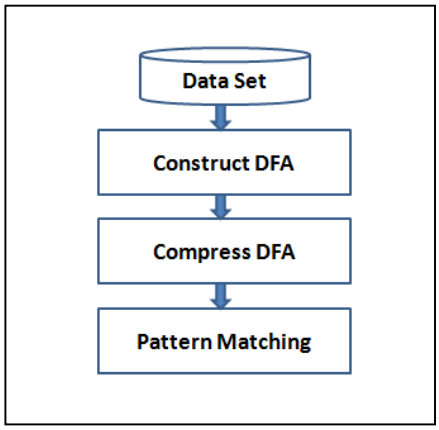

Figure 1.DFA compression system for efficient pattern matching

3.1. DFA construction

The Aho-Corasick algorithm [2] is used for DFA construction and pattern matching at initial stage. This DFA is generated from regular expression. First regular expression is converted into NFA [13] and then NFA is converted into DFA [14]. The Aho-Corasick state machine constructed for patterns set {EBC, EBBC, BA, BBA, BCD, CF} [1] is shown in Figure 2. In that state machine states s_3, s_5, s_7, s_9, s_{11}, s_{13} are the final states of patterns EBC, EBBC, BA, BBA, BCD, and CF respectively.

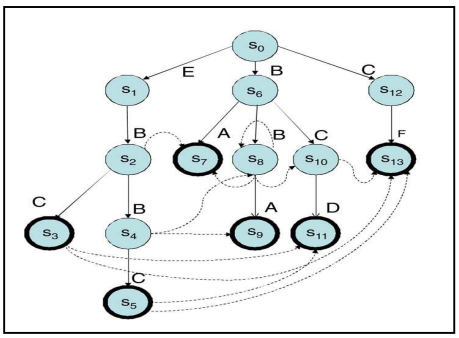

Figure 2. DFA

3.2. DFA compression

The compression approach uses CompactDFA [1] algorithm to compress the DFA. This algorithm lessens the rule set to only one rule for each state. A single prefix is used to resolve all transitions to a particular state. A DFA is compressed by algorithm and then it is stored in to memory. In pattern matching process, this compressed DFA is used. The output of compression process is set of compressed rules for DFA. It stores the set of rules with current state field, symbol field and next state field [1]. A DFA compression algorithm works in three phases. DFA states are grouped in state grouping phase. Later on common suffix tree is constructed in common suffix tree construction phase and finally state and rule encoding phase produces compressed storage rules for DFA [1].

In the state grouping phase, a group of the states in DFA are created based on Common Suffix (CS) and Longest Common Suffix (LCS) parameters [1]. For every state in DFA, these two parameters are calculated. If there are more than one incoming transitions for a state, then common suffix parameter is a label of the state without its last symbol [1]. Otherwise, there is no common suffix parameter is calculated for that state. The longest common suffix for state is long length common suffix of a state to which state s has an outgoing edge [1]. The state grouping stage takes DFA as input and produces set of CS and LCS as an output.

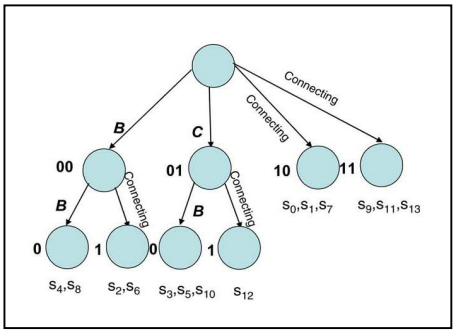

Figure 3. Common suffix tree

In the common suffix tree creation phase, common suffix tree is created and suffix rules are converted in to prefix rules. The nodes in common suffix tree are different LCS values called as set L. For every two values l_1, l_2 in set L, l_1 is an ancestor of l_2 if and only if l_1 is a suffix of l_2 [1].

Connecting nodes are added for every internal node. At the end, states are linked to one of the connecting nodes. The Common suffix tree stage takes set of CS and LCS and builds common suffix tree as an output. The common suffix tree [1] generated from DFA is shown in Figure 3.
In the state and rule encoding phase, there is a procedure of encoding of common suffix tree, states and rules. First code width is calculated which is a number of bits required to encode the common suffix tree. Then edges and nodes are encoded. At the end, using corresponding node in common suffix tree every state is encoded. The state and rule encoding algorithm [1] is used to produce compressed rules. This stage takes common suffix tree and produces compressed rule set for DFA. In pattern matching process, these compressed rules are used. The rules of compressed DFA [1] are shown in Figure 4.

	Current State	Symbol	Next State
1	00000	C	01001 (s_5)
2	00100	C	01000 (s_3)
3	00100	B	00000 (s_4)
4	10010	B	00100 (s_2)
5	010**	D	11010 (s_{11})
6	000**	A	11000 (s_9)
7	01***	F	11100 (s_{13})
8	00***	C	01010 (s_{10})
9	00***	B	00010 (s_8)
10	00***	A	10100 (s_7)
11	*****	E	10010 (s_1)
12	*****	C	01100 (s_{12})
13	*****	B	00110 (s_6)
14	*****	*	10000 (s_0)

Figure 4. Rules of compressed DFA

3.3 Pattern matching

The pattern matching process uses rules of compressed DFA on input data. In compressed set of rules, a state code can match multiple rules at a time. In that case rule with longest prefix match is selected [1]. For example, if input data contains "CF" string then pattern matching process start at roots node s_0. At state s_0 on input symbol 'C', current state field is any match and next state field is s_{12} from rule 12 in Figure 4. Therefore, transition to state s_{12} is taken and now s_{12} is the current state. Next, at state s_{12} with its code 01100 on input symbol 'F', rule 7 is matched. In rule 7, next state is s_{13} which is the final state hence string "CF" is accepted by DFA. It concludes that given input text data has signatures of pattern of interests.

IV. EXPERIMENTAL RESULTS

To evaluate performance of compression approach, we used Snort virus data set [12]. We have tested the performance of uncompressed and compressed DFA with parameters as storage space of DFA in bytes on simple pattern data set without using regular expression. Table 1 shows partial results of an efficient DFA compression system.

Table 1. DFA Size Comparison between uncompressed DFA and compressed DFA on pattern dataset

No. of Patterns	Size of uncompressed DFA	Size of compressed DFA
100	20344 bytes	3368 bytes
1000	100652 bytes	9920 bytes
5000	202556 bytes	11394 bytes

CONCLUSION AND FUTUREWORK

Storage space of deterministic finite automata and time required for pattern matching are important issues in pattern matching process. Traditional approach for storing deterministic finite automata required more memory space. It affects on time required for pattern matching. It is very important to reduce the storage space of deterministic finite automata to make pattern matching process fast. The approach described in this paper uses bit reduction method to compress Aho-Corasick deterministic finite automata generated from regular expression. All transitions to particular state are represented by single prefix using state grouping and common suffix tree methods. Compressed rule set are generated by state and rule encoding method. The pattern matching process traverses the compressed form of deterministic finite automata to check existence of patterns in the input data. As a future work, one may create and compress deterministic finite automata that can detect malicious code encoded in special characters for security purpose.

REFERENCES

[1] AnatBremler-Barr, D.Hay, Y. Koral, "CompactDFA:Scalable pattern matching Using Longest Prefix Match Solutions," in IEEE/ACM Transaction on networking,vol-22,No.2,April 2014.

[2] A.V. Aho and M.J. Corasick. "Efficient String Matching: An Aid to Bibliographic Search." Communications of the ACM, 18(6):333–340, 1975.

[3] S. Kumar, S. Dharmapurikar, F. Yu, P. Crowley, and J. Turner, "Algorithms to accelerate multiple regular expressions matching for deep packet inspection", in Proc. of ACM SIGCOMM , pages 339-350. ACM, 2006.

[4] S. Kumar, J. Turner, J. Williams, "Advanced algorithms for fast and scalable deep packet inspection", in Proc. ACM/IEEE Symp. Archit. Netw. Commun. Syst. (ANCS), pages 81-92. ACM, 2006.

[5] M. Becchi, P. Crowley, "A hybrid finite automaton for practical deep packet inspection", in Proc. Conf. Emerging Netw. Exp. Technol.(CoNEXT), pages 1-12, 2007.

[6] S. Kumar, B. Chandrasekaran, J. Turner, G. Varghese, "Curing regular expressions matching algorithms from insomnia, amnesia, and acalculia", in Proc. ACM/IEEE Symp. Archit. Netw. Commun. Syst. (ANCS), pages 155-164. ACM, 2007.

[7] R. Smith, C. Estan, and S. Jha, "Xfa: Faster signature matching with extended automata", in IEEE Symposium on Security and Privacy, May 2008.

[8] D.Ficara, S.Giordano, G. Procissi, F.Vitucci, G.Antichi, A.D. Pietro, "An Improved DFA for Fast Regular Expression Matching" ACM SIGCOMM Computer Communication Review, Volume 38, Number 5, October 2008.

[9] S. Kong, R. Smith, and C. Estan, "Efficient signature matching with multiple alphabet compression tables," in Proc. Int. Conf. Security Privacy Commun. Netw. (Securecomm), 2008.

[10] S. Kumar, J. Turner, P. Crowley, and M. Mitzenmacher, "HEXA: Compact data structures for faster packet processing", in Proc. IEEE Conf. Comput. Commun. (INFOCOM), 2009.

[11] L. Tan, and T. Sherwood, "A High Throughput String Matching Architecture for Intrusion Detection and Prevention," ISCA 2005

[12] "SNORT," 2010 [Online]. Available: http://www.snort.org

[13] Ken Thompson, "Programming techniques: regular expression search algorithm", in Communications of the ACM, Pages 419-422 , Volume 11 Issue 6, June 1968

[14] John E. Hopcroft and Jeffrey D. Ullman, "Introduction to Automata Theory, Languages, and Computation", Addison-Wesley Publishing, Reading Massachusetts, 1979.

