Computers and Electrical Engineering 67 (2018) 223–234

Contents lists available at ScienceDirect

Computers and Electrical Engineering

journal homepage: www.elsevier.com/locate/compeleceng

Intelligent and efficient grouping algorithms for large-scale regular expressions☆

Zhe Fu*[a,b], Kai Wang[d], Liangwei Cai[e], Jun Li[b,c]

[a] Department of Automation, Tsinghua University, Beijing, China
[b] Research Institute of Information Technology (RIIT), Tsinghua University, Beijing, China
[c] Tsinghua National Laboratory for Information Science and Technology (TNList), Beijing, China
[d] Yunshan Networks, Beijing, China
[e] College of Information Engineering, Shenzhen University, Shenzhen, Guangdong, China

ARTICLE INFO

Keywords:
Regular expression matching
Grouping algorithms
DFA
Intelligent optimization

ABSTRACT

Regular expressions are widely used in various applications. Due to its low time complexity and stable performance, Deterministic Finite Automaton (DFA) has become the first choice to perform fast regular expression matching. Unfortunately, compiling a large set of regular expressions into one DFA often leads to the well-known state explosion problem, and consequently requires huge memory consumption. Regular expression grouping is a practical approach to mitigate this problem. However, existing grouping algorithms are either brute-force or locally optimal and thus not efficient for large-scale regular expressions. In this paper, we propose two grouping algorithms, namely Reevo and Reant, to solve this problem intelligently and efficiently. In addition, two optimization methods are presented to accelerate the convergence speed and reduce the running time of proposed algorithms. Experimental results on large-scale rulesets from real world show that our algorithms achieve around 25% to 45% improvement compared to previous regular expression grouping algorithms.

1. Introduction

Nowadays, pattern matching is playing important roles in many practical applications. In network security, the payloads of network packets are scanned against a set of predefined patterns to identify the potential security threats, including viruses, intrusions, spams, data leakage and so forth. Bioinformatic research utilizes pattern matching to search for the similarity of DNA/RNA sequences. As the patterns are getting more and more complex, exact strings are being substituted gradually by regular expressions [1], which have more expressiveness and flexibility. To perform regular expression matching, regular expressions are compiled into Nondeterministic Finite Automaton (NFA) or Deterministic Finite Automaton (DFA). NFA has fewer states and transitions, and its space cost linearly depends on the size of regular expression set. Thus NFA is space-efficient. However, in the worst case, there can be massive state traversals per input character for NFA, leading to tremendous memory bandwidth requirement and slow matching speed. On the contrary, DFA only activates one state and requires precisely one state traversal per input character. As a result, DFA is fast, and thus it is the preferred choice for time-sensitive applications such as deep inspection in network security.

However, the time-efficiency of DFA is at the cost of the huge amount of memory consumption. Compared to NFA, DFA needs far more states and transitions, and suffers from the *state explosion* problem in many cases. What's more, the number of regular

☆ Reviews processed and recommended for publication to the Editor-in-Chief by Associate Editor Dr. Y. Sang.

* Corresponding author.
E-mail addresses: fu-z13@mails.tsinghua.edu.cn (Z. Fu), wangkai@yunshan.net.cn (K. Wang), cailw@szu.edu.cn (L. Cai), junl@tsinghua.edu.cn (J. Li).

https://doi.org/10.1016/j.compeleceng.2018.03.040
Received 30 June 2017; Received in revised form 23 March 2018; Accepted 23 March 2018
Available online 30 March 2018
0045-7906/

expressions in practical use is increasing rapidly. The open-source network intrusion prevention system Snort [2] has already employed more than 27,000 rules[1] that are written in regular expressions. Such a large scale of regular expressions make it impossible to build a single DFA from these regular expressions on commodity processing platforms. Yu et al. [3] summarized several categories of regular expressions that can cause blow-up of corresponding DFA, and one major factor is concluded as the interactions among regular expressions. In general, for the regular expressions with interactions, the DFA constructed from them has a far larger size than the sum of the sizes of DFAs which are built from each regular expression individually. Based on this analysis, a number of regular expression grouping algorithms have been put forward to address this problem. However, the most state-of-the-art methods [3–6] can hardly find the optimum solution, and the techniques they employ do not fit for large-scale regular expressions.

In this paper, we propose novel grouping algorithms for regular expressions to solve the *state explosion* problem and reduce the memory consumption when regular expressions are constructed into DFAs. In detail, our contributions can be concluded as follows. First, we analyze the practical demands of regular expression grouping and formulate the grouping problem into two separate aspects: (1) when the number of groups is specified, minimize the total number of DFA states and (2) when the memory space of each group is limited, minimize the number of groups. Then, two intelligent and efficient grouping algorithms, namely Reevo and Reant, are proposed for each goal. In addition, we put forward two optimizations, including an improved approach to approximating the number of DFA states and a method to obtain better initial solutions, in order to accelerate the convergence speed and reduce the running time of the proposed algorithms. Experimental results based on practical large-scale rulesets show that our grouping algorithms could obtain around 25%–45% improvement compared to previous grouping algorithms.

This paper is organized as follows. Section 2 states the related work about regular expression matching. In Section 3, we illustrate the motivation of regular expression grouping and categorize the grouping problem into two kinds of situations based on practical demands. Two grouping algorithms are proposed correspondingly in Section 4. In Section 5, further optimizations for our grouping algorithms are put forward. Experimental results are shown in Section 6. In the last section, we give our conclusion.

2. Related work

During the past decades, many techniques for improving regular expression matching have been proposed. Most studies are aimed at compressing the transitions or states of DFA to reduce the memory consumption. Kumar et al. [7] invented D^2FA (Delayed input DFA) to reduce the number of transitions by adding a default transition, and it diminishes the memory occupied by DFAs to represent the regular expressions. However, it is at the expense of more memory accesses overhead, because multiple default transitions will be executed when no corresponding transitions are found in the compressed states for certain input character. The D^2FA-derived algorithms (like A-DFA [8] *etc*.) are all designed for further improving the matching speed or compression ratio. PaCC [9] partitions a complex regular expression into multiple simple segments without semantic overlap, and uses a Relation Mapping Table (RMT) to record their dependencies. But this method only works for several types of complex regular expressions. Other techniques for reducing DFAs' space consumption such as state merging [10], character set reduction [11] similarly reduce memory at the cost of more matching time and worse temporal efficiency. Our methods are orthogonal and complementary to these methods.

Several studies leverage hardware platform to improve the executing efficiency of regular expression matching. FPGA based methods [12,13] have advantages in the implementation with pipeline and parallelism. However, the small size of on-chip memories of FPGA limits the practical deployment of large-scale rulesets. GPU based methods [14] take advantage of the massive parallel execution units and high memory bandwidth, while the performance is sensitive to the divergences in memory accesses and execution path. TCAM devices are fast [15], but the updating algorithm may cost large amounts of storage space. Worse more, the high cost and big power consumption make TCAM devices not cost-effective for large-scale regular expression matching. The regular expression grouping methods can help obtain more efficient data layout on these hardware platforms to benefit maximally from hardware acceleration.

Regular expression grouping is a relatively new and independent direction to solve the *state explosion* problem of DFA. In [3], Yu et al. first put forward the ideas of greedily grouping regular expressions to construct multiple automata, and implemented a grouping algorithm that only uses the information whether a regular expression interacts with each other, without quantifying the interaction relationship. Moreover, the simplicity of this algorithm makes it run into local optimum easily. Becchi et al. [16] simply divided the rulesets binarily until the DFA states number of each group is smaller than a given limit. Rohrer et al. [4] evaluated the *distance* between each pair of regular expressions and converted the grouping problem to the Maximum Cut problem. Methods including Poles Heuristic and Simulated Annealing are leveraged based on the *distance* relationships among regular expressions. RegexGrouper [5] estimates the DFA size in a similar way by measuring the convolvement relationship. The authors map the k-grouping problem to the maximum k-cut problem in theory, and prove the grouping solution is not less than $1 - k^{-1}$ times of the optimal partition of the corresponding maximum k-cut problem. Other proposals are based on these ideas and get similar results [6]. However, all of these algorithms only find sub-optimal solutions within a reasonable running time, and could hardly deal with various situations where regular expressions are grouped.

[1] Extracted from snortrules-snapshot-29110, which was released on January 4, 2018.