**湖 南 科 技 大 学**

**毕 业 设 计（ 论 文 ）**

|  |  |
| --- | --- |
| **题目** | 软件定义网络的  字符串匹配压缩算法 |
| **作者** | 游 芊 |
| **学院** | 计算机科学与工程学院 |
| **专业** | 网络工程 |
| **学号** | 1805020234 |
| **指导教师** | 唐 兵 |

二〇二二年 五 月 二十八 日

**湖 南 科 技 大 学**

**毕业设计（论文）任务书**

计算机科学与工程 院 网络工程 系（教研室）

系（教研室）主任: （签名） 年 月 日

**学生姓名:** 游芊 **学号:** 1805020234 **专业:** 网络工程

1 设计（论文）题目及专题： 软件定义网络的字符串匹配压缩算法

2 学生设计（论文）时间：自 2022 年 1 月 17 日开始至 2022 年 5 月 28 日止

3 设计（论文）所用资源和参考资料：

（1）Aho A V, Corasick M J. Efficient string matching: an aid to bibliographic search[J]. Communications of the ACM, 1975, 18(6): 333-340.

（2）Bremler-Barr A, Hay D, Koral Y. CompactDFA: Generic state machine compression for scalable pattern matching[C]//2010 Proceedings IEEE INFOCOM. IEEE, 2010: 1-9.

（3）Yun S K. An efficient TCAM-based implementation of multipattern matching using covered state encoding[J]. IEEE Transactions on Computers, 2010, 61(2): 213-221.

4 设计（论文）应完成的主要内容：

（1）实现现有的基于公共后缀迁移边的状态编码算法和覆盖状态编码算法；

（2）提出一种基于默认迁移边的状态编码算法；

（3）进行算法设计实现与评估。

5 提交设计（论文）形式（设计说明与图纸或论文等）及要求：

（1） 提交毕业论文；

（2） 提交程序源代码；

（3） 将毕业论文电子档、程序源代码、答辩PPT刻录成光盘并提交。

6 发题时间： 2022 年 1 月 11 日

指导教师： （签名）

学 生： （签名）

**湖 南 科 技 大 学**

**毕业设计（论文）指导人评语**

[主要对学生毕业设计（论文）的工作态度，研究内容与方法，工作量，文献应用，创新性，实用性，科学性，文本（图纸）规范程度，存在的不足等进行综合评价]

**指导人：** （签名）

年 月 日

**指导人评定成绩：**

**湖 南 科 技 大 学**

**毕业设计（论文）评阅人评语**

[主要对学生毕业设计（论文）的文本格式、图纸规范程度，工作量，研究内容与方法，实用性与科学性，结论和存在的不足等进行综合评价]

**评阅人： （签名）**

**2022年 5 月 24 日**

**评阅人评定成绩：**

**湖 南 科 技 大 学**

**毕业设计（论文）答辩记录**

**日期：** 2022年5月26日

**学生：** 游芊 **学号：** 1805020234 **班级：** 网络工程二班

**题目：** 软件定义网络的字符串匹配压缩算法

**提交毕业设计（论文）答辩委员会下列材料：**

**1 设计说明书/ 论 文 共 页**

**2 设计（论文）图 纸 共 页**

**3 指导人、评阅人评语 共 页**

**毕业设计（论文）答辩委员会评语：**

[主要对学生毕业设计（论文）的研究思路，设计（论文）质量，文本图纸规范程度和对设计（论文）的介绍，回答问题情况等进行综合评价]

**答辩委员会主任：** （签名）

**委员：** （签名）

（签名）

（签名）

（签名）

**答辩成绩：**

**总评成绩：**

**摘 要**

软件定义网络（Software Defined Network , SDN）通常使用高速的三态内容可寻址存储器（Ternary Content Addressable Memory , TCAM）来存储数据报头信息。受防火墙、应用流量识别、入侵检测系统等应用需求的驱动，SDN需要对数据报内容进行处理，因此研究基于TCAM的字符串匹配算法非常重要。

Aho-Corasick算法将字符串集构建成一个确定性的有限自动机(Deterministic Finite Automata , DFA)，给定一个当前状态和一个输入字符，当前状态就会迁移到确定的目的状态，因此每两个状态和一个输入字符就可以构成一条字符串规则。如果将所有状态迁移存入TCAM表中，会造成很大的TCAM表存储开销。已有的基于公共后缀迁移边的状态编码算法（Common Suffix Transitions-based State Encoding , CSTSE）通过构建公共后缀树，将每个状态所对应的字符串规则编码为公共后缀迁移边，大大减少了TCAM表项数，但其编码宽度过宽，造成TCAM表的存储开销和能耗过大。现有的覆盖状态编码算法（Covered code State Encoding , CSE）利用DFA中的失效状态来对状态进行编码，CSE的压缩效果不好，TCAM表项过大。

本文提出了一种基于默认迁移边的状态编码算法（Default Transitions-based State Encoding , DTSE），利用了默认迁移边这一思想构建一颗默认迁移树，然后进行状态编码、迁移边融合。实验评估结果表明，DTSE压缩之后的TCAM表项数和CSTSE的TCAM表项数一样达到了最优，且编码宽度比CSTSE小一个数量级，DTSE大大降低了TCAM表的存储开销，提高了TCAM表的匹配吞吐量。

关键词：字符串匹配算法；TCAM；DFA；Aho-Corasick算法

**ABSTRACT**

Software Defined Network (SDN) usually uses high-speed Ternary Content Addressable Memory (TCAM) to store data header information. Driven by the application requirements of firewalls, application traffic identification, intrusion detection systems, etc., SDN needs to process the content of datagrams, so it is very important to study the string matching algorithm based on TCAM.

The Aho-Corasick algorithm builds the set of strings into a Deterministic Finite Automata (DFA), given a current state and an input character, the current state will migrate to a deterministic destination state, so every two state and an input character can form a string rule. If all state transitions are stored in the TCAM table, it will cause a large storage overhead of the TCAM table. The existing Common Suffix Transitions-based State Encoding algorithm (CSTSE) based on common suffix transition edges constructs a common suffix tree and encodes the string rules corresponding to each state as common suffix transition edges, which greatly reduces TCAM entries, but its encoding width is too wide, resulting in excessive storage overhead and energy consumption of the TCAM table. The existing Covered Code State Encoding algorithm (CSE) uses the failure state in the DFA to encode the state, the compression effect of CSE is not good, and the TCAM table entry is too large.

This paper proposes a Default Transitions-based State Encoding algorithm (DTSE), which uses the idea of default transition edges to construct a default transition tree, and then performs state encoding and transition edge fusion. The experimental evaluation results show that the number of TCAM entries after DTSE compression is the same as the number of TCAM entries of CSTSE, and the encoding width is an order of magnitude smaller than that of CSTSE. DTSE greatly reduces the storage overhead of TCAM tables and improves TCAM tables. matching throughput.

**Keywords:** String matching; TCAM; DFA; Aho-Corasick algorithm

目 录

[第一章 绪论 1](#_Toc979)

[1.1 研究背景 1](#_Toc5675)

[1.2 国内外的研究现状 1](#_Toc4550)

[1.3 论文的结构安排 2](#_Toc18561)

[1.4 小结 3](#_Toc9054)

[第二章 相关理论知识 4](#_Toc25227)

[2.1 Aho-Corasick字符串匹配算法 4](#_Toc3814)

[2.2 基于TCAM的字符串匹配算法 6](#_Toc157)

[2.2.1 TCAM结构与匹配原理 6](#_Toc27086)

[2.2.2 基于TCAM的字符串匹配算法 6](#_Toc19243)

[2.3 小结 7](#_Toc2163)

[第三章 基于TCAM的字符串匹配压缩算法 8](#_Toc17391)

[3.1 基于公共后缀迁移边的状态编码算法 8](#_Toc12594)

[3.1.1 算法思想和算法设计 8](#_Toc23856)

[3.1.2 算法实现 12](#_Toc17678)

[3.1.3 算法时间复杂度 12](#_Toc13405)

[3.2 覆盖状态编码算法 12](#_Toc29288)

[3.2.1 算法思想与算法设计 12](#_Toc21076)

[3.2.2 算法实现 17](#_Toc27292)

[3.2.3 算法时间复杂度 18](#_Toc9997)

[3.3 小结 18](#_Toc6541)

[第四章 基于默认迁移边的状态编码算法 19](#_Toc25480)

[4.1 状态编码算法设计 19](#_Toc5716)

[4.2 算法实现 24](#_Toc11303)

[4.3 算法复杂度分析 24](#_Toc32439)

[4.4 小结 25](#_Toc11997)

[第五章 实验评估 26](#_Toc15126)

[5.1 实验环境 26](#_Toc17372)

[5.2 实验结果 27](#_Toc22247)

[5.2.1 TCAM表项数 27](#_Toc13476)

[5.2.2 TCAM表存储开销 28](#_Toc14847)

[5.2.3 TCAM表构建时间 29](#_Toc25767)

[5.2.4 TCAM匹配吞吐量 31](#_Toc12601)

[5.2.5 TCAM能耗 31](#_Toc12244)

[5.3 小结 32](#_Toc26938)

[第六章 总结与展望 34](#_Toc14034)

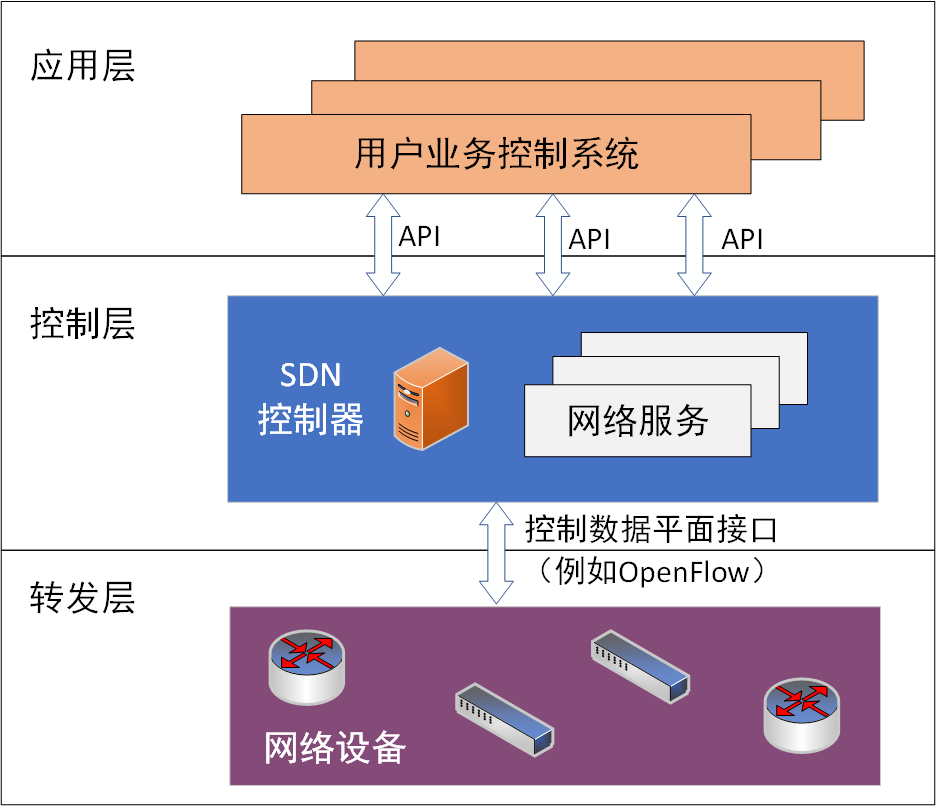
[参考文献 35](#_Toc21598)

[致谢 36](#_Toc19885)

第一章 绪论

1.1 研究背景

网络应用技术发展飞快，一种新兴的网络技术软件定义网络（Software Defined Network , SDN）被提出。SDN将控制平面、数据平面解耦，仅通过一个通用接口就能够控制整个分布式网络，非常灵活。图1.1是SDN的基本架构图，从图中可以看出，SDN有转发层、控制层、应用层3层。转发层的网络设备用于负责进行数据包的转发；网络协议通常由控制层执行，控制层和转发层之间依赖控制数据平面接口协议，比如OpenFlow协议；应用层就是用户业务控制系统，可以进行远程监控。



**图1.1 SDN的基本架构图**

SDN通常用于对数据报头信息进行处理，而不能处理数据报内容。SDN中有多个分布式交换机，进行数据包分类[1][2][3][4]的研究，其中使用三态内容可寻址存储器（Ternary Content Addressable Memory , TCAM）用来存储IP五元组。

字符串匹配在防火墙、应用流量识别、入侵检测系统等网络应用中存在应用需求，用来检测数据报内容中是否存在可疑的内容。首先从字符串规则集中的解析出字符串集，然后把解析出来的字符串集构建成一个DFA，每两个状态和一个输入字符构成一条字符串规则。TCAM也可以用来存储字符串规则，将所有的状态迁移存入TCAM表，并行搜索所有TCAM表项，进行字符串匹配。但当前的入侵检测系统需要处理大规模的字符串规则集，如果直接存入所有的字符串规则，避免不了会造成存储开销过大，且TCAM存储容量受限，价格也非常昂贵，因此，研究基于TCAM的字符串匹配压缩算法非常必要。

1.2 国内外的研究现状

接下来介绍SDN中的高速字符串匹配算法的研究现状。字符串匹配算法有基于软件和硬件两个方面的研究，而基于硬件的解决方案主要分为两种类别：基于FPGA/ASIC的和基于TCAM的。本文主要对基于TCAM的字符串匹配压缩算法进行研究。

字符串匹配问题最早的解决方案是由Aho和Corasick提出的Aho-Corasick算法，简称AC算法[5]。AC算法主要将输入的字符串集构建成一个树形的有限状态机，首先通过构建goto迁移和Failure迁移来构造一个不确定性的有限自动机（Non-deterministic Finite State Automata , NFA），再将不确定的有限自动机转化成为一个确定的有限自动机，AC算法细节详见本文2.1节。

然而，由于直接存储字符串集会造成很大的存储开销和浪费，研究者们提出了各种字符串匹配压缩算法[6-9]。Sailesh Kumer等人提出了一种能够大大减少DFA中状态之间迁移边的数量的方案，D2FA算法[6]，基于构建默认迁移边来实现，算法时间复杂度为O(n2logn)。Michela Beecchi对D2FA算法进行了优化，提出了时间复杂度为O(n2)的默认迁移边构造算法[7]。A Bremler-Barr等人提出了基于公共后缀迁移边的状态编码算法[8]，利用所有状态的传入状态都共享一个公共后缀这一特性来构建TCAM表项，这可以有效压缩TCAM表项数，但其编码宽度过宽，导致存储空间过大。SangKyun Yun等人提出了覆盖状态编码算法[9]，利用TCAM三态特性和状态压缩[11]思想，根据失效状态来构建一颗失效树，然后对源状态进行覆盖状态编码，压缩TCAM的表项数，但其压缩后的TCAM表项数仍然很多，TCAM能耗问题依旧得不到解决。

1.3 论文的结构安排

针对上述研究所存在的问题，本文对降低TCAM表存储开销和能耗问题做了研究，提出了一种基于默认迁移边的状态编码算法。

本文共有6个章节，每个章节的内容概括如下：

第一章：介绍字符串匹配压缩算法的研究现状。

第二章：阐述最早的Aho-Corasick字符串匹配算法和基于TCAM的字符串匹配算法，介绍了TCAM的结构和TCAM的匹配原理。

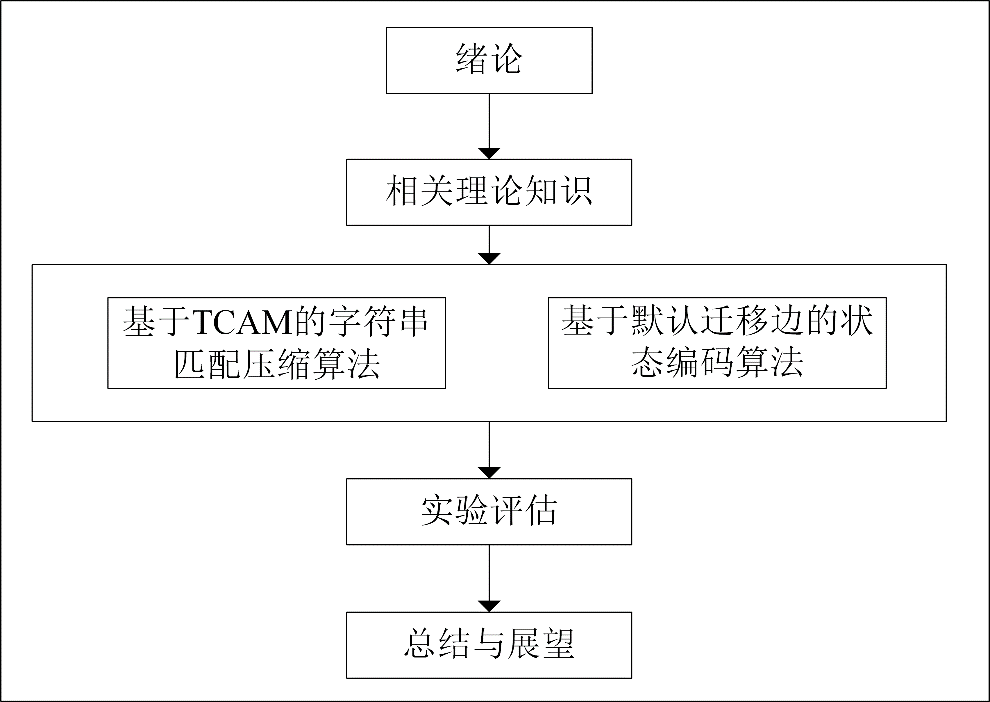
第三章：介绍现有的基于公共后缀迁移边的状态编码算法（Common Suffix Transitions-based State Encoding algorithm , CSTSE）和覆盖状态编码算法（Covered Code State Encoding algorithm , CSE）的算法思想与算法设计。

第四章：提出一种基于默认迁移边的状态编码算法（Default Transitions-based State Encoding algorithm , DTSE）。

第五章：从TCAM表项数、TCAM表存储开销、TCAM表构建时间、TCAM匹配吞吐量、TCAM能耗这五个指标对CSTSE、CSE、DTSE三个算法进行实验评估。

第六章：总结全文，分析工作中所存在的不足之处。

各章节的结构关系如下图1.2。



**图1.2 章节结构图**

1.4 小结

本章介绍了国内外的研究背景，阐述了国内外针对字符串匹配算法的研究现状，概括了本文的论文结构安排。

第二章 相关理论知识

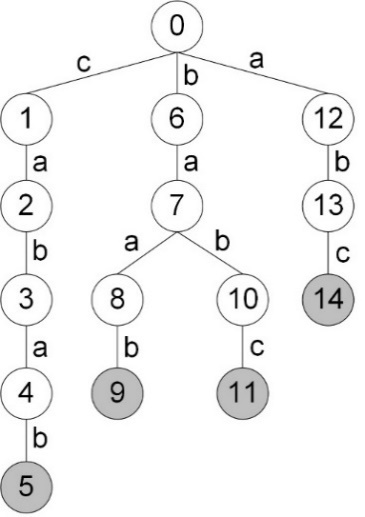
所谓字符串匹配，就是在一个待匹配的字符串中查找其中是否出现一个给定的字符串集中的字符串。本章介绍两种最常见的字符串匹配算法，分别是Aho-Corasick字符串匹配算法和基于TCAM的字符串匹配算法。

2.1 Aho-Corasick字符串匹配算法

Aho-Corasick算法，简称AC算法，主要是以线性速度解决多个字符串的匹配问题。传统暴力匹配也可以解决多字符串的匹配问题，但是采取暴力匹配解决的时间复杂度过高，而AC算法将字符串匹配转换成状态转移，即AC算法通过构建一个树形有限自动机，用确定性的有限自动机中的各个状态之间的迁移来表示字符串匹配过程，AC算法只需要扫描一次待匹配的字符串就可以得到匹配结果，时间复杂度上优于传统暴力匹配。

AC算法的过程：用给定的字符串集构建一个树形DFA，然后把待匹配的字符串放入DFA中，在各个状态之间进行转换，当跳转到某些特定的终结状态时，记录下匹配到的字符串。

AC算法包括构建DFA和匹配字符串两个过程。构建DFA包括构建goto迁移和Failure迁移。以字符串集{cabab , baab , babc , abc}为例，首先我们构建goto迁移。goto迁移的构建过程是每读入一个字符串，从初始状态开始依次读入字符，若通过该字符有可到达的状态则直接进行状态迁移，否则创建一个新状态，图2.1是字符串集{cabab , baab , babc , abc}的状态迁移图。

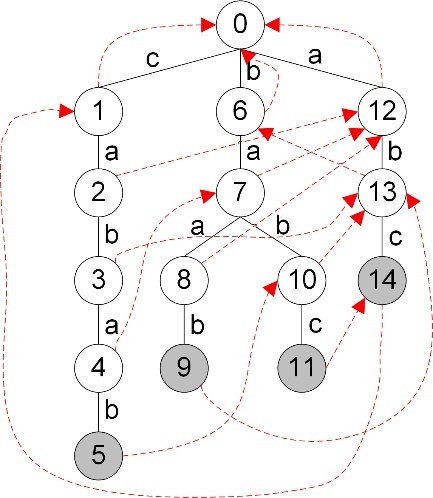


**图2.1 状态迁移图**

在DFA中，初始状态为状态0。当读入字符串cabab，状态0首先查找通过字符c是否有可到达的状态，若没有，就会新建一个状态1，状态0通过字符c就到达新的状态1。以此类推，当读入字符串cabab中的最后面的一个字符b，当前状态就会迁移到新的状态5。此时，字符串cabab读取完毕，将该字符串的最后一个字符所到达的状态，即状态5，标记为终结状态。然后读入下一个字符串baab，此时从初始状态0开始，状态0通过字符b无可到达的状态，则依次读入baab创建新状态，标识状态9为终结状态。

读入字符串babc，和前两个不太一样，从状态0读入字符b状态迁移到状态6，从状态6读入字符a可到达状态7，不需要创建新状态，但状态7通过字符b没有可到达的状态，则创建新状态10，最后读入字符c，创建状态11并标记为终结状态。最后一个字符串abc以此类推。

在进行字符串匹配的过程中，若在当前状态读入一个字符无可到达的状态，Failure迁移就派上用场了。图2.2是构建完成Failure迁移之后的NFA，构建过程如下。首先，第一层状态的Failure迁移默认指向初始状态0。以状态2为例分析其Failure状态，因为其父节点通过字符a到达状态2，则找状态1的失效状态0通过字符a是否存在可到达的状态，状态0通过字符a到达状态12，即状态2的Failure状态为状态12。



**图2.2 NFA**

那如何进行字符串匹配呢？以待匹配字符串abca为例，依次读入abc，初始状态0迁移到状态14，这个时候，状态14被标记为终结状态，则字符串abc匹配成功。读入最后一个字符a，但状态14通过字符a无可到达状态，则找其Failure状态1，状态1通过字符a到达状态2，状态2未被标记为终结状态，字符串遍历结束，匹配到一个字符串abc。

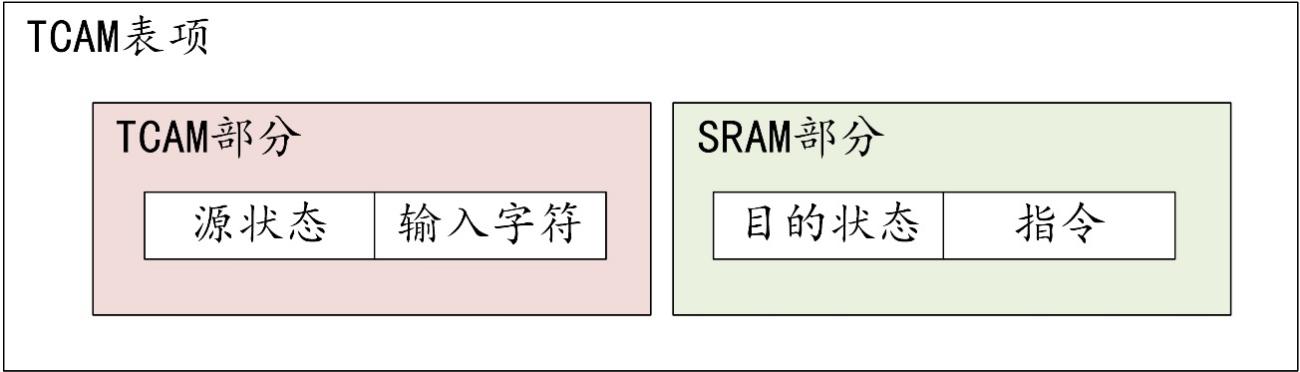
AC算法的时间复杂度为O(nlogn)。构建有限自动机包括构建goto迁移和构建Failure迁移。构建goto迁移只需要遍历输入的字符串集，则时间复杂度为O(n)；构建Failure迁移因其需要沿着父状态及其失效状态往上遍历，所以时间复杂度为O(nlogn)。就AC算法的匹配过程而言，只需要遍历一次待匹配的字符串就可以得到匹配结果，显而易见，时间复杂度为O(n)。

2.2 基于TCAM的字符串匹配算法

2.2.1 TCAM结构与匹配原理

三态内容可寻址存储器（TCAM）是高速处理器，能够快速完成并行查找，即可以在一个时钟周期内并行搜索所有表项，返回匹配结果。TCAM拥有三态特性，可以表示“0”、“1”、“\*”三种状态，其中“\*”表示既可以匹配“0”，也可以匹配到“1”。

图2.3是TCAM的结构图。每个TCAM表项在物理组成上都有两部分，即一个TCAM部分和一个SRAM部分。TCAM部分存储源状态和输入字符，SRAM部分存储对应的目的状态和附加的指令，这些指令在TCAM进行匹配时作为查找结果返回。



**图2.3 TCAM结构图**

TCAM和SRAM一起合作，TCAM输出匹配到的表项地址，而SRAM存储匹配到的数据。匹配电路将当前状态、输入字符连接起来，形成一个搜索键，然后将该搜索键并行地与所有TCAM表项进行比较。若搜索键匹配，TCAM输出第一个匹配表项的索引，然后将SRAM中相同索引的值，即目的状态和附加的指令提供给逻辑电路，更新寄存器和当前状态。

2.2.2 基于TCAM的字符串匹配算法

基于TCAM的字符串匹配算法就是将DFA中所有状态迁移存入TCAM表。最常见的有两种存放方式，一种是存放理论DFA，一种是存放实际DFA。理论DFA即把各个状态通过字符表上256个字符的迁移边全部存入TCAM中，而实际DFA则只用将各个状态通过字符串集中出现过的字符的迁移边存储在TCAM中，将其他未出现过的字符每个只用占一个TCAM表项。

以字符串集{cabab , baab , babc , abc}为例，一共有15个状态。

将理论DFA存入TCAM表，则有15\*256=3840条迁移边，TCAM的表项数等于迁移边数，当规则集很大的时候，显然对应的TCAM表项数会很多，所以实际应用并不采用理论DFA，而是采用实际DFA的解决方案，因为该字符串集中只出现了abc三个字符，则只用存储15\*3+253=298条迁移边，较理论DFA相比，实际DFA能够有效压缩TCAM的表项数。TCAM表部分展示如表2.1。

**表2.1 TCAM表**

|  |  |  |
| --- | --- | --- |
| **源状态** | **输入字符** | **目的状态** |
| 0 | a | 12 |
| 1 | a | 2 |
| 2 | a | 0 |
| 3 | a | 4 |
| … | … | … |

如何在TCAM表中进行字符串匹配呢？例如当前状态为3，输入字符a，首先将3和a组成一个搜索键，在TCAM部分进行查找，将查找到的TCAM表项地址返回给SRAM部分，SRAM部分根据地址返回目的状态12。

2.3 小结

本章介绍了Aho-Corasick字符串匹配算法和基于TCAM的字符串匹配算法。以字符串集{cabab , baab , babc , abc}和待匹配的字符串abca为例具体地阐述了AC算法过程。然后具体介绍了TCAM的基本结构和匹配原理，提到了常见的两种存储DFA的方法，即存储实际DFA和存储理论DFA。

第三章 基于TCAM的字符串匹配压缩算法

本章介绍两种已有的字符串匹配压缩算法，分别是基于公共后缀迁移边的状态编码算法（CSTSE）和覆盖状态编码算法（CSE）。CSTSE基于所有到特定状态节点的迁移都可以用定义成一组公共前缀这一观察，然后将字符串匹配问题简化为现已研究非常充分的最长前缀匹配(Longest Prefix Match , LPM)问题。CSE是通过构造一棵失效树，利用TCAM的三态特性对源状态进行编码，以便一个编码能够覆盖其他的编码，从而达到压缩TCAM表项数的效果。

3.1 基于公共后缀迁移边的状态编码算法

3.1.1 算法思想和算法设计

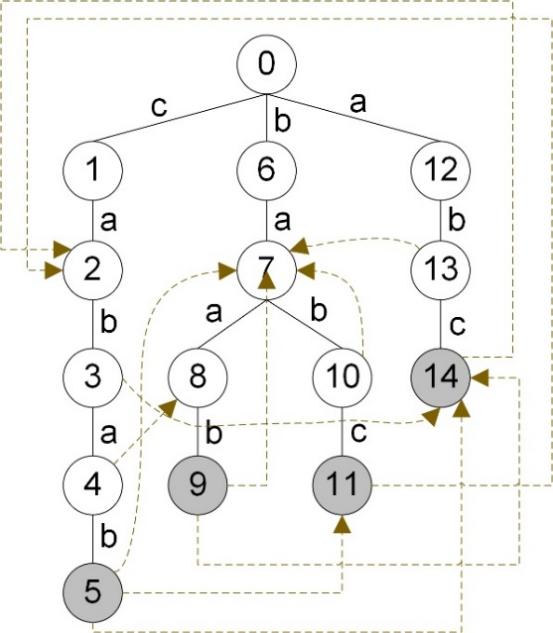
CSTSE是由Anat Bremler-Barr等人通过观察，将字符串匹配问题化简成IP查找问题，虽然这两个问题看起来是不相关的，但是，CSTSE将每条字符串规则定义为一个使用了公共前缀的状态集，当一个状态可以匹配多条规则时，选择带有最长公共前缀的规则，与IP查找中的逻辑完全相同。

CSTSE有以下三个步骤：

**步骤1：**计算公共后缀（Common Suffix , CS）和最长公共后缀（Longest Common Suffix , LCS）；

**步骤2：**构建公共后缀树；

**步骤3：**对状态节点和规则进行编码。



**图3.1 DFA**

以字符串集{cabab , baab , babc , abc}为例，首先，我们构建一个DFA，如图3.1所示。

虚线代表交叉迁移，图中省略了指向第一层节点的迁移边。这个交叉迁移是在失效路径（通过AC算法得到的失效状态及其父节点，不包含初始状态0）的基础上，若状态s通过字符ch到达初始状态s0，而其失效路径上的节点s1通过字符ch到达下一状态s2，则状态s指向s2。

例如，在图3.1中，状态4的失效路径是状态7和状态12，状态7通过字符a到达状态8，而状态4通过字符a到达初始状态0，则状态4指向状态8，因状态4通过字符b到达非初始状态5，所以不指向状态7通过字符b所到达的下一状态10，以此类推，状态4也不指向状态12通过字符b所到达的下一状态13。

接下来对CSTSE的步骤进行详细介绍。

**（1）计算CS和LCS**

假设每个状态s用一个标签来进行编码，状态s的传入邻居是N(s)。我们将会发现，在N(s)中，所有状态的标签都共用一个共同的后缀，这个后缀是不包括传入状态s的最后一个字符。这样，通过将code(N(s))作为没有最后一个符号的标签，用“\*”填充它，然后再应用最长前缀匹配规则，就能准确地捕捉到DFA迁移。

如何计算每个状态的CS和LCS呢？

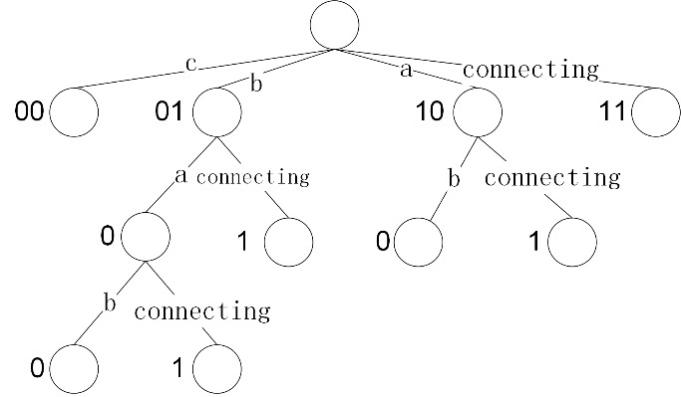
如果一个状态s只有一条传入边，则状态s的CS值为空，即CS(s)= 。否则，状态s的CS则为状态s的传入边所带的标签的共同的后缀，其中标签不包括迁移到状态s的那个字符。

例如，状态2的传入状态有3个状态，即N(s)={1,11,14}。状态1的标签的c，状态11的标签的babc，状态14的标签的abc，所以状态2的公共后缀为c，即CS(2)=c。

计算完所有状态的CS，再计算所有状态的LCS，一个状态的LCS主要是找该状态节点输出边所指向的状态中最长的CS。

例如，状态5的输出边有3条，分别传出到7、11、14这三个状态，而CS(7)=b，CS(11)=bab，CS(14)=ab，就可以得到状态5的最长公共后缀是bab，即LCS(5)=bab。

**（2）构建公共后缀树**

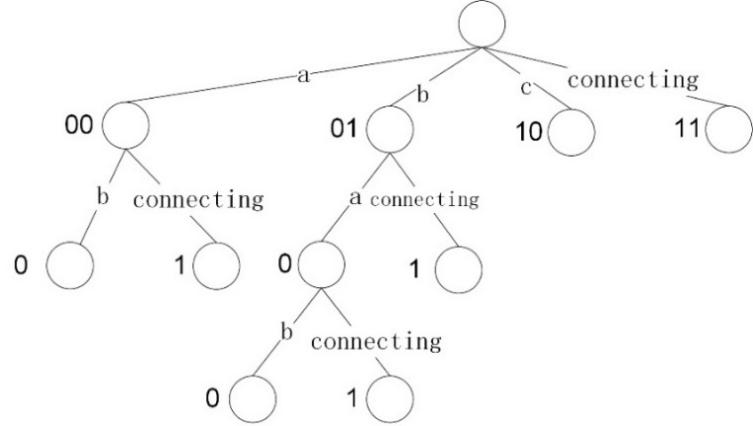


**图3.2 公共后缀树**

将所有状态的最长公共后缀字符串反转过来构成一个最长公共前缀集，然后将最长公共前缀构建成一棵树。

接着找到编码宽度w，它是编码整个公共后缀树所需要的比特位数。对于公共后缀树的每个非叶子节点s都会添加至少一条connecting边使得节点s的子节点数目是2n。

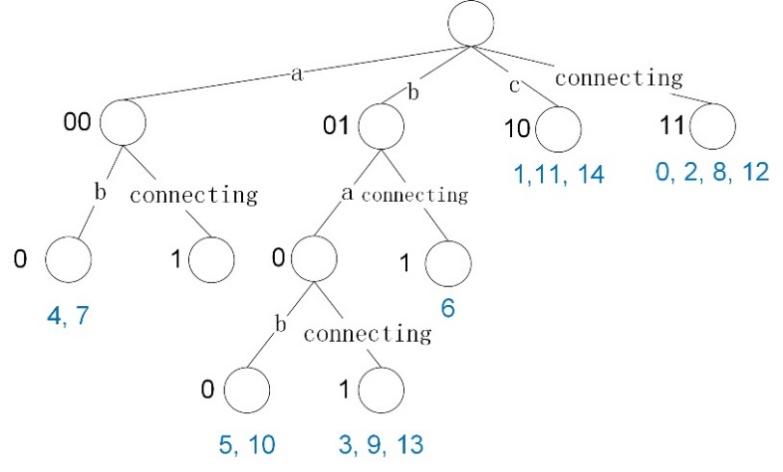
图3.2是CSTSE的公共后缀树。接下来枚举来自同一父节点的所有兄弟边来编码，这个编码宽度d=logn，其中n为该状态的父节点的孩子节点数量，图3.3表示带有编码的公共后缀树。



**图3.3 带有编码的公共后缀树**

**（3）对状态节点和规则进行编码**

根据每个状态s的最长公共后缀LCS，将其分配到公共后缀树的叶子节点。 如图3.4所示。

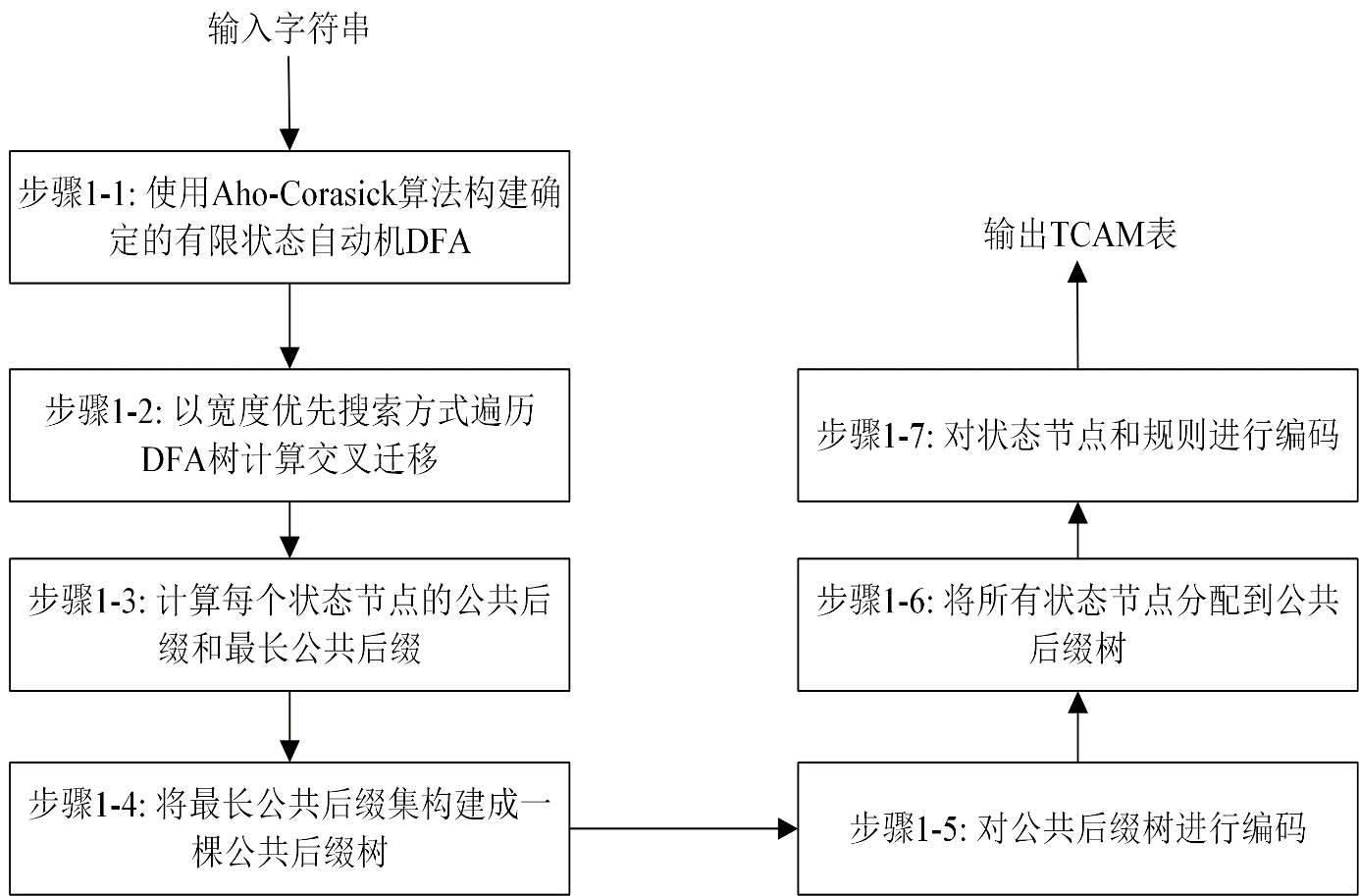


**图3.4 将状态节点分配到公共后缀树**

比如LCS(1)=c，LCS(11)=c，LCS(14)=c，所以状态1、11、14这三个状态分配到第一层的第三个节点。状态6的最长公共后缀是b，因为通过字符b到达的第一层第二个结点不是叶子节点，所以再往下走一层，到达一个由connecting边连接起来的叶子节点。

**表3.1 CSTSE的状态编码表**

|  |  |  |  |
| --- | --- | --- | --- |
| **状态** | **公共后缀** | **源状态编码** | **目的状态编码** |
| 0 | NULL | \*\*\*\*\*\* | 110000 |
| 1 | NULL | \*\*\*\*\*\* | 100000 |
| 2 | c | 10\*\*\*\* | 110001 |
| 3 | 2 | 110001 | 010100 |
| 4 | 3 | 010100 | 000000 |
| 5 | 4 | 000000 | 010000 |
| 6 | NULL | \*\*\*\*\*\* | 011000 |
| 7 | b | 01\*\*\*\* | 000001 |
| 8 | ba | 000\*\*\* | 110010 |
| 9 | 8 | 110010 | 010101 |
| 10 | 7 | 000001 | 010001 |
| 11 | bab | 0100\*\* | 100001 |
| 12 | NULL | \*\*\*\*\*\* | 110011 |
| 13 | 12 | 110011 | 010110 |
| 14 | ab | 010\*\*\* | 100010 |



**图3.5 CSTSE算法流程图**

然后根据每个状态所在的叶子节点为其分配目的状态编码，目的状态编码会填充“0”以达到整棵树的编码宽度w。

源状态编码根据状态的公共后缀进行编码，源状态编码都会填充“\*”以达到整棵公共后缀树的编码宽度w。

如状态14的公共后缀是ab，则其源状态是010\*\*\*，因为其公共后缀ab反转过来的路径是010，又因为整棵公共后缀树的编码宽度w为6，则填充“\*”。

最后，CSTSE的状态编码如下表3.1所示。

3.1.2 算法实现

CSTSE算法的实现流程图如图3.5所示。

3.1.3 算法时间复杂度

使用基于公共后缀迁移边的状态编码算法中的时间复杂度为O(nlogn)，其中n是状态节点的数量，具体分析如下。

计算CS、LCS需要依赖于状态节点的失效路径，所以时间复杂度是O(nlogn)，即步骤1的时间复杂度为O(nlogn)。构建公共后缀树的时间复杂度是O(nlogn)，即树遍历算法的时间复杂度。对状态节点和规则进行编码只需要对每个节点分配编码，又因为进行源状态编码的时候需要遍历公共后缀树，所以步骤3的时间复杂度为O(nlogn)。

3.2 覆盖状态编码算法

3.2.1 算法思想与算法设计

CSE利用TCAM的三态特性进行编码状态，使得一种编码可以覆盖其它编码。例如，编码\*\*\*\*涵盖了全部的4比特编码，而编码11\*\*包含了4个编码1100、1101、1110和1111。由于SRAM中的目的状态字段不能存储“\*”，因此 SRAM中的编码必须是“0”或“1”。然而，TCAM中的当前状态字段可以存储“\*”，所以TCAM中可以存放覆盖编码。

CSE算法包括以下四个阶段：

**步骤1：**构造一颗失效树；

**步骤2：**确定每个状态的编码维度和大小；

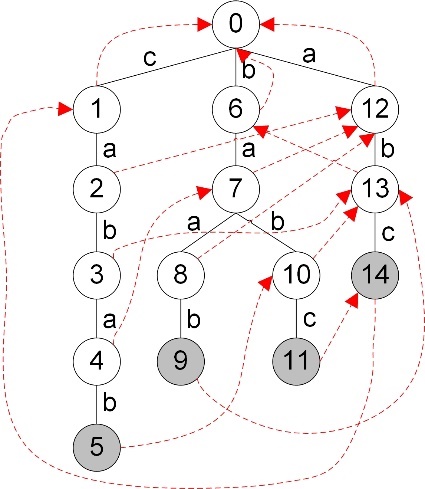
**步骤3：**为每个状态分配源状态编码和目的状态编码；

**步骤4：**构建TCAM表项。

以字符串集{cabab , baab , babc , abc}为例，构建NFA，如图3.6。

**（1）构造一棵失效树**

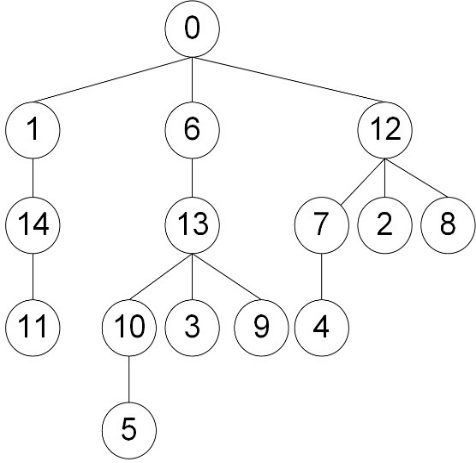
失效树是由在NFA中每个状态把其失效状态作为父节点构成的一棵树，初始状态0成为失效树的根节点。而失效状态的计算根据AC算法中构建失效迁移这一步骤，即可计算出每个状态的失效状态。失效树如图3.7所示。



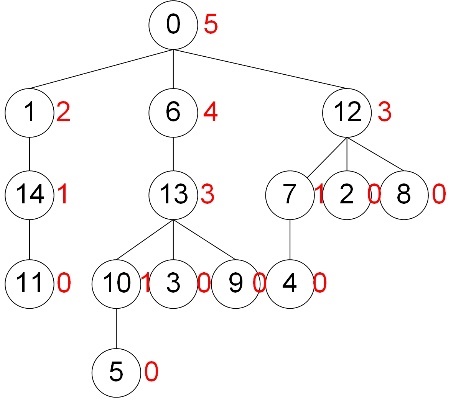
**图3.6 NFA**

**（2）确定每个状态节点的编码维度和大小**

源状态编码中的“\*”位数称为它的维度，而源状态中的包含的覆盖编码数量称为它的大小。一个状态s的源状态编码的维度和大小分别用dim(s)和size(s)表示。



**图3.7 失效树**



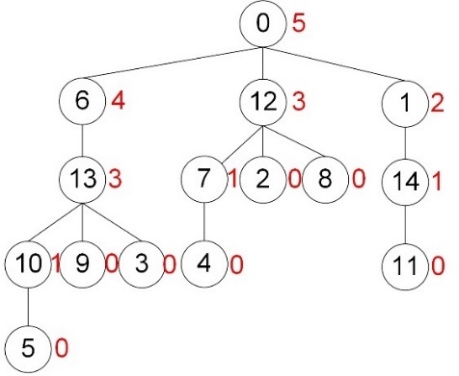
**图3.8 带有维度的失效树**

在失效树中，叶子节点的维度为0，因为叶子节点不需要覆盖其他任何状态，它的源状态编码和目的状态编码是一样的。对于其他非叶子节点，它会包含它自己本身和它后代全部的状态节点。因此，状态节点s的维度可以用以下公式来计算：



图3.8是计算好维度的失效树。

**（3）状态编码**



**图3.9 排序后的失效树**

首先，需要对失效树的节点的所有子节点按照编码维度大小降序的顺序排序，排序后的失效树如图3.9。

接下来，自顶向下分配源状态编码和目的状态编码。

首先，根据上面状态节点维度计算公式计算出初始状态0的编码宽度w为5，为初始状态0分配源状态编码“\*\*\*\*\*”，目的状态“00000”。由图3.8可知，状态6的编码大小为4，则为其分配源状态编码“1\*\*\*\*”，为其分配目的状态编码“10000”；状态6的子节点13的编码长度为3，则在状态6的源状态编码基础上为其分配源状态编码“11\*\*\*”，目的状态编码则为“11000”。继续为状态10、9、3分配编码，状态10的源状态编码为“1111\*”，目的状态编码为“11110”，对其兄弟状态9分配源状态编码“11101”，因为源状态编码不包含“\*”，所以状态9的目的状态编码也为“11101”，同理，可推导出状态3的源状态编码和目的状态编码都是“11100”。

对于一个状态节点，如果其源状态编码有包含“\*”，则其目的状态编码使用“0”替代“\*”。按照以上方式，可以得出所有状态节点的源状态编码和目的状态编码（见下表3.2）。

**表3.2 状态编码表**

|  |  |  |
| --- | --- | --- |
| **状态** | **源状态编码** | **目的状态编码** |
| 0 | \*\*\*\*\* | 00000 |
| 1 | 001\*\* | 00100 |
| 2 | 01101 | 01101 |
| 3 | 11100 | 11100 |
| 4 | 01111 | 01111 |
| 5 | 11111 | 11111 |
| 6 | 1\*\*\*\* | 10000 |
| 7 | 0111\* | 01110 |
| 8 | 01100 | 01100 |
| 9 | 11101 | 11101 |
| 10 | 1111\* | 11110 |
| 11 | 00111 | 00111 |
| 12 | 01\*\*\* | 01000 |
| 13 | 11\*\*\* | 11000 |
| 14 | 0011\* | 00110 |

**（4）构建TCAM表项**

将所有状态通过输入字符到达不同的目的状态来构建TCAM表项，同时，我们需要进行迁移边融合，即尽可能合并通过相同字符到达相同目的状态的源状态节点。

以字符a为例，表3.3是字符a相关的所有TCAM表项。但可以发现，状态节点2、4、8、12之间存在父子关系，状态节点2、4、8、12都通过字符a到达目的状态0，则可以将这四条迁移边合并成一条源状态编码为01\*\*\*目的状态编码为00000的迁移边。

**表3.3 压缩前的TCAM表**

|  |  |  |
| --- | --- | --- |
| **源状态** | **输入字符** | **目的状态** |
| \*\*\*\*\* | a | 01000 |
| 001\*\* | a | 01101 |
| 01101 | a | 00000 |
| 11100 | a | 01111 |
| 01111 | a | 00000 |
| 11111 | a | 00000 |
| 1\*\*\*\* | a | 01110 |
| 0111\* | a | 01100 |
| 01100 | a | 00000 |
| 11101 | a | 00000 |
| 1111\* | a | 00000 |
| 00111 | a | 00000 |
| 01\*\*\* | a | 00000 |
| 11\*\*\* | a | 00000 |
| 0011\* | a | 00000 |

完成父子节点迁移边融合之后，我们发现还有迁移边之间的只有一位不同源状态编码通过相同字符到达相同目的状态，我们将这类迁移边进行融合，成为兄弟节点迁移边融合。

**表3.4 父子节点融合后的TCAM表**

|  |  |  |
| --- | --- | --- |
| **源状态** | **输入字符** | **目的状态** |
| \*\*\*\*\* | a | 01000 |
| 001\*\* | a | 01101 |
| 01\*\*\* | a | 00000 |
| 11100 | a | 01111 |
| 11\*\*\* | a | 00000 |
| 1\*\*\*\* | a | 01110 |
| 0111\* | a | 01100 |
| 0011\* | a | 00000 |

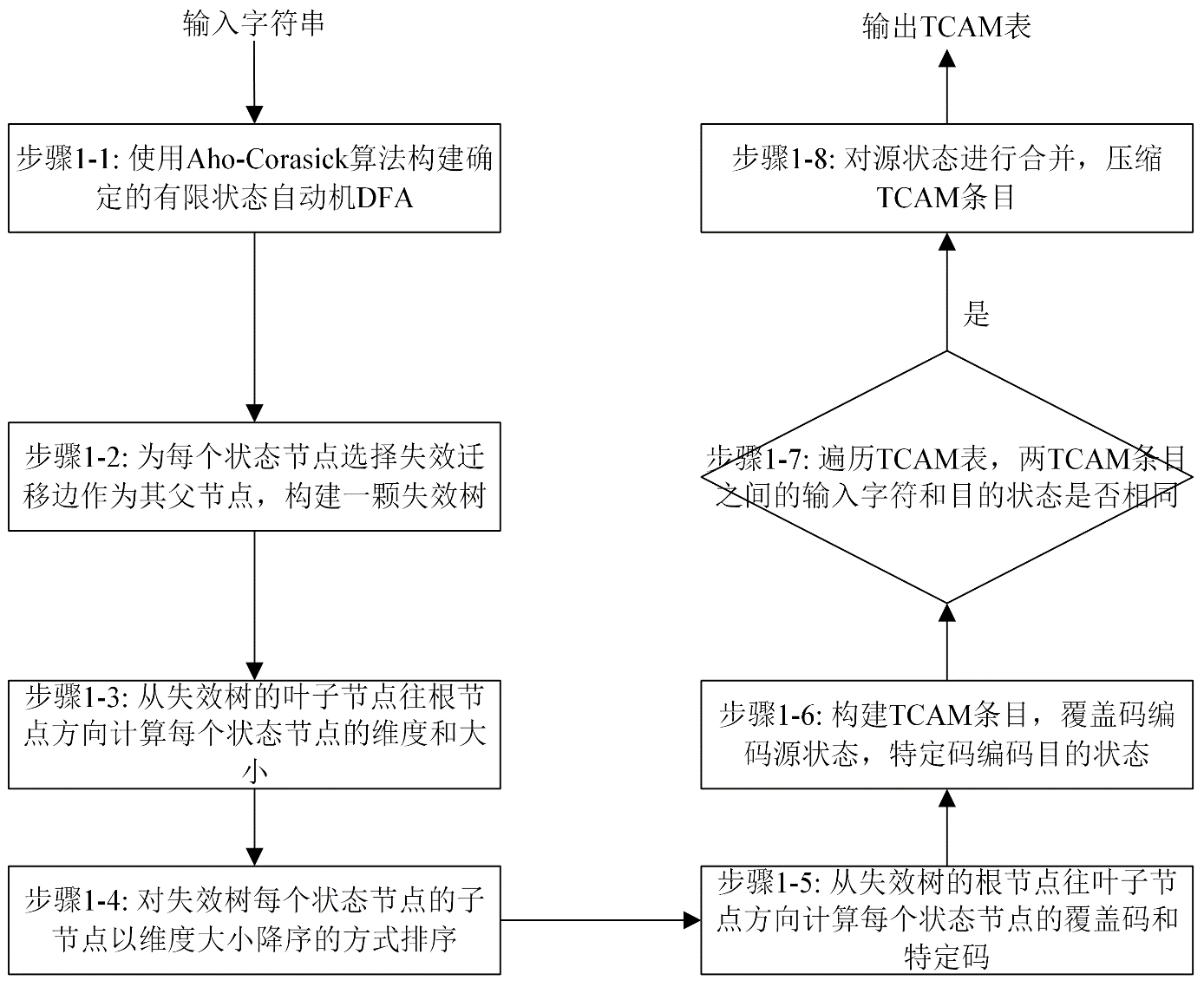
在表3.4中，我们发现源状态编码分别为01\*\*\* 和11\*\*\*通过字符a到达相同目的状态0。则我们可以将其融合成一条迁移边，源状态编码用01\*\*\*来表示。最终得到的迁移边压缩成表3.5。

**表3.5 压缩后的TCAM表**

|  |  |  |
| --- | --- | --- |
| **源状态** | **输入字符** | **目的状态** |
| \*\*\*\*\* | a | 01000 |
| 001\*\* | a | 01101 |
| 01\*\*\* | a | 00000 |
| 11100 | a | 01111 |
| 1\*\*\*\* | a | 01110 |
| 0111\* | a | 01100 |
| 0011\* | a | 00000 |

3.2.2 算法实现

接下来展示CSE算法流程图。



**图3.10 CSE算法流程图**

3.2.3 算法时间复杂度

CSE算法由四个步骤按顺序执行，步骤4迁移边融合的时间复杂度起决定性作用。前3个步骤进行状态编码的时间复杂度为O(nlogn)，因为其实现依赖于失效树。步骤4进行迁移边融合，时间复杂度为O(n2)。

3.3 小结

本章介绍了CSTSE和CSE两个已有算法。

CSTSE的主要优点之一是它适用于商业上的IP查找解决方案，这也就意味着CSTSE也可用于执行快速的字符串匹配。虽然CSTSE压缩后的迁移边数非常少，但其编码宽度由最长的那个字符串决定，在大规则数据集中，对应的最长的字符串长度会很长，则编码宽度非常大，导致其存储开销过大，TCAM能耗过高。且CSTSE算法的实现比较复杂。

CSE虽然其在构建失效树的时间复杂度不高，且在编码宽度上表现良好，算法简单易懂，但其压缩后的迁移边数过多，压缩效果不好。

第四章 基于默认迁移边的状态编码算法

4.1 状态编码算法设计

**表4.1 默认迁移边分析表**

|  |  |
| --- | --- |
| **状态** | **祖先节点列表** |
| 0 | **0(0,256)** |
| 1 | **0(0,255)** |
| 2 | **0(0,255)**、1(1,254)、6(1,254)、*12(1,255)* |
| 3 | 0(0,254)、1(1,254)、6(1,254)、12(1,253)、2(2,253)、7(2,253)、**13(2,255)** |
| 4 | 0(0,254)、1(1,254)、6(1,254)、12(1,254)、2(2,254)、**7(2,255)**、13(2,253)、3(3,253)、8(3,254)、10(3,253)、14(2,254) |
| 5 | 0(0,254)、1(1,254)、6(1,255)、12(1,253)、2(2,253)、7(2,253)、13(2,255)、3(3,254)、8(3,253)、**10(3,256)**、14(2,254)、4(4,253)、9(4,255)、11(4,254) |
| 6 | **0(0,255)** |
| 7 | **0(0,254)**、*1(1,254)*、*6(1,254)、12(1,254)* |
| 8 | **0(0,255)**、1(1,254)、6(1,254)、*12(1,255)*、*2(2,255)*、7(2,254)、13(2,253) |
| 9 | 0(0,254)、1(1,254)、6(1,255)、12(1,253)、2(2,253)、7(2,253)、**13(2,256)**、3(3,255)、8(3,253)、10(3,255)、14(2,254) |
| 10 | 0(0,254)、1(1,254)、**6(1,255)**、12(1,253)、2(2,253)、7(2,253)、*13(2,255)* |
| 11 | 0(0,255)、**1(1,256)**、6(1,255)、12(1,254)、2(2,254)、7(2,254)、13(2,254)、3(3,254)、8(3,254)、10(3,254)、*14(2,256)* |
| 12 | **0(0,255)** |
| 13 | 0(0,254)、1(1,254)、**6(1,255)**、12(1,253) |
| 14 | 0(0,255)、**1(1,256)**、6(1,255)、12(1,254)、2(2,254)、7(2,254)、13(2,254) |

基于有限自动机中的状态是没有特定实际含义的这一观察，我们进行源状态编码和目的状态编码。通过状态压缩这一思想，我们将多条迁移边合并到单个TCAM表项中。我们为中的每个状态分配源状态编码和目的状态编码来替换它的状态编号。源状态编码由“0”、“1”或“\*”组成的三元编码，它可以用来表示多个源状态，目的状态编码是由“0”和“1”组成的特定编码，用来某一确定的目的状态。

源状态编码可使用三元编码，而目的状态编码不能使用三元编码的原因是源状态存储在TCAM表项的TCAM部分，TCAM支持三态，但目的状态存储在TCAM表项的SRAM部分，SRAM不支持三态。

基于默认迁移边的状态编码算法包括以下4个步骤：

**步骤1：**构造默认迁移树；

**步骤2：**计算每个状态节点编码宽度；

**步骤3：**进行源状态编码和目的状态编码；

**步骤4：**迁移边融合。

**（1）构造默认迁移树**

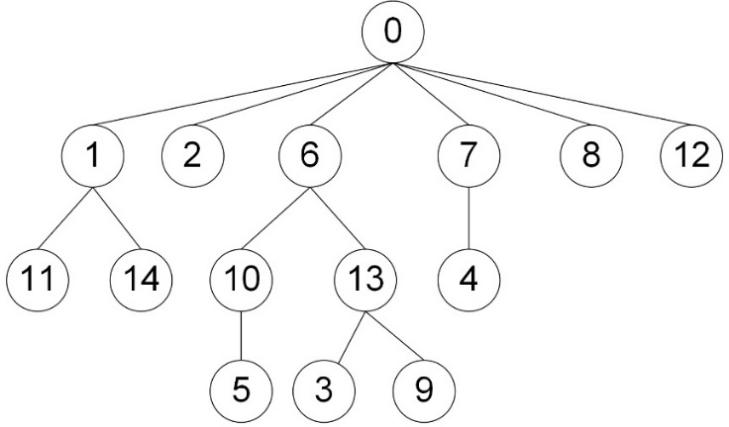
我们利用D2FA的默认迁移边来构造一颗默认迁移树。默认迁移树的边表示两状态之间的默认迁移边。

那怎么计算两状态之间的默认迁移边呢？

每个状态与层次低的祖先节点比较通过相同字符到达相同目的状态的迁移边数，找出层次最低的默认迁移边数最多的状态作为其父节点。表4.1展示出所有状态的默认迁移边的分析表，左边表示状态，右边表示状态的祖先节点列表，其中0(0,256)表示状态0与状态0的默认迁移边数是256，且状态节点0的深度为0。

以状态2为例对其默认迁移边进行分析，状态2会与所有祖先状态节点相比较，但深度比状态2小的状态有状态0、1、6、12。其中，与状态2具有相同默认迁移边数最多的是状态0和状态12，有255条，但因为状态0的深度小于状态12的深度，所以状态2选择状态0作为其默认迁移。

依上类推，可以得到一颗默认迁移树，如图4.1。



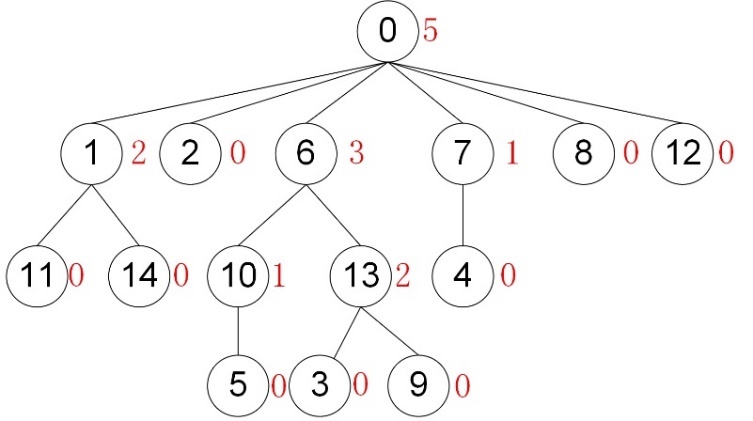
**图4.1 默认迁移树**

**（2）计算每个状态的编码宽度**

编码宽度表示源状态编码中 “\*” 的位数。我们以自底向上的方式计算每个状态节点的编码宽度。状态s的编码宽度与其后代状态的数量有关。每个叶子状态的编码宽度设置为0，然后每个祖先状态的编码宽度递归计算如下：



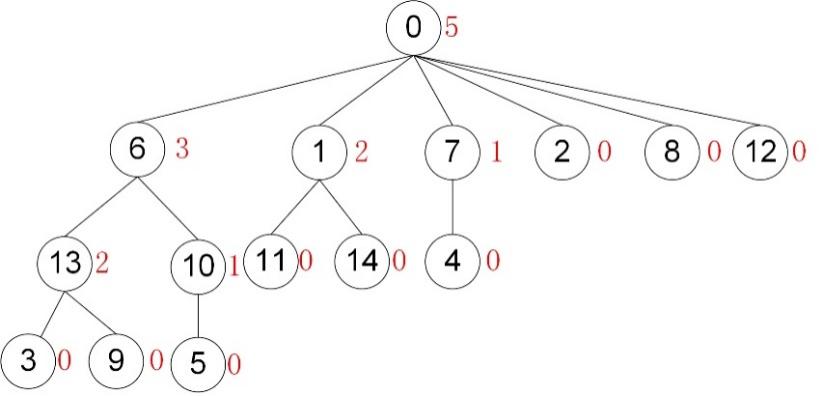
根据上述公式计算出所有状态的编码宽度，如图4.2。



**图4.2 带有编码宽度的默认迁移树**

**（3）进行源状态编码和目的状态编码**

在进行源状态编码和目的状态编码之前，我们需要对默认迁移树的所有状态节点以编码宽度降序进行排序。图4.3展示了排序后的默认迁移树，其中圆圈旁边的数字代表该状态的编码宽度。



**图4.3 排序之后的默认迁移树**

由于状态节点0的编码宽度为5，所以整棵默认迁移树的编码宽度为5。则状态节点0的源状态编码为“\*\*\*\*\*”，目的状态编码为“00000”。状态节点6的编码宽度为3，则源状态编码为“11\*\*\*”，目的状态编码为“11000”。

**表4.2 状态编码表**

|  |  |  |
| --- | --- | --- |
| **状态** | **源状态编码** | **目的状态编码** |
| 0 | \*\*\*\*\* | 00000 |
| 1 | 101\*\* | 10100 |
| 2 | 10001 | 10001 |
| 3 | 11111 | 11111 |
| 4 | 10011 | 10011 |
| 5 | 11011 | 11011 |
| 6 | 11\*\*\* | 11000 |
| 7 | 1001\* | 10010 |
| 8 | 10000 | 10000 |
| 9 | 11110 | 11110 |
| 10 | 1101\* | 11010 |
| 11 | 10111 | 10111 |
| 12 | 01111 | 01111 |
| 13 | 111\*\* | 11100 |
| 14 | 10110 | 10110 |

**表4.3 压缩前的TCAM表**

|  |  |  |
| --- | --- | --- |
| **源状态** | **输入字符** | **目的状态** |
| \*\*\*\*\* | a | 01111 |
| 101\*\* | a | 10001 |
| 10001 | a | 01111 |
| 11111 | a | 10011 |
| 10011 | a | 10000 |
| 11011 | a | 10010 |
| 11\*\*\* | a | 10010 |
| 1001\* | a | 10000 |
| 10000 | a | 01111 |
| 11110 | a | 10010 |
| 1101\* | a | 10010 |
| 10111 | a | 10001 |
| 01111 | a | 01111 |
| 111\*\* | a | 10010 |
| 10110 | a | 10001 |

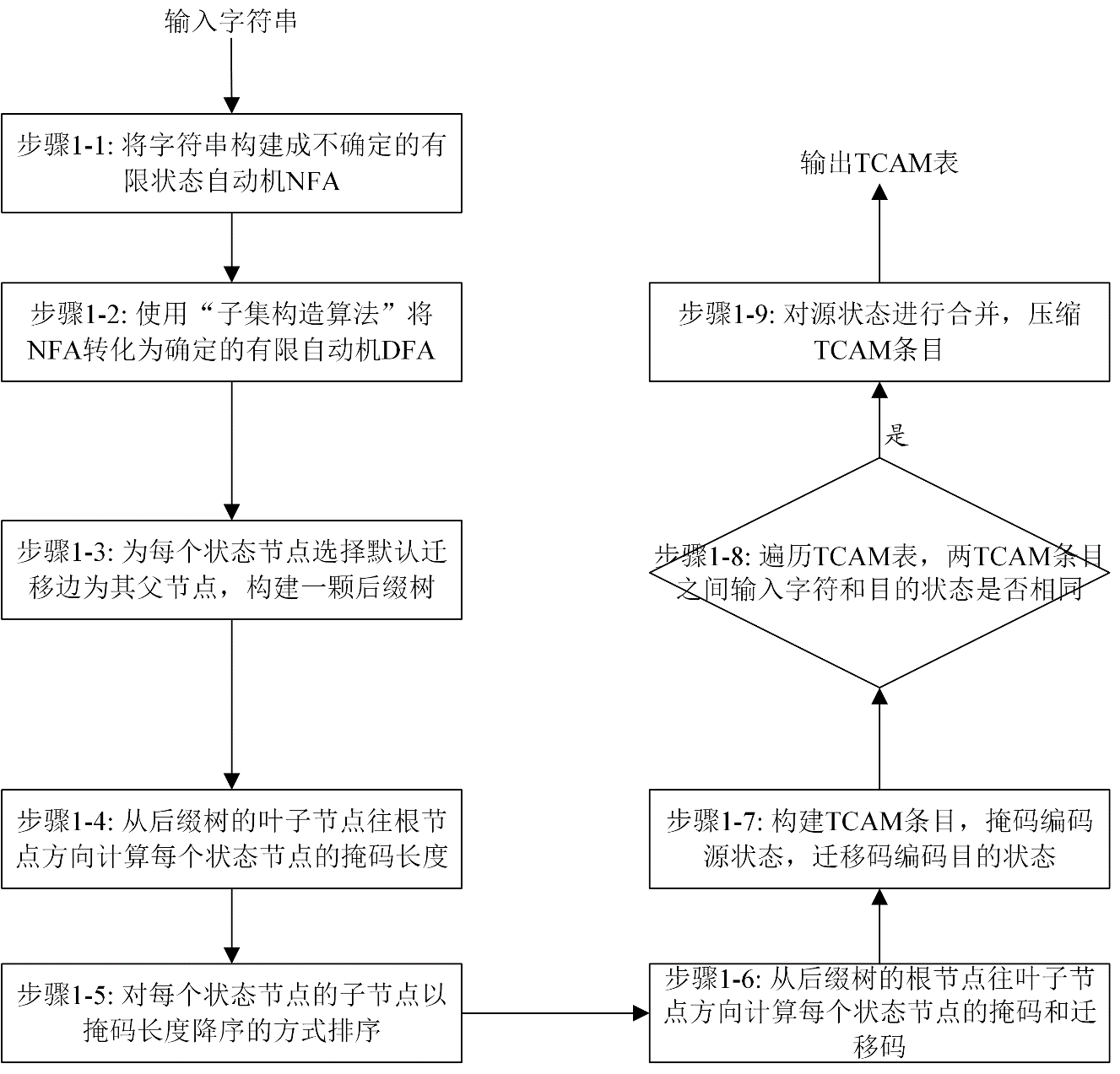
DTSE分配编码方式和CSE分配编码方式的思想一样，在3.2.2小节阐述清楚了如何分配源状态编码和目的状态编码。详见3.2.2小节。DTSE的状态编码表见表4.2。

**（4）迁移边融合**

因为源状态编码是由“0” “1” “\*”表示的，所以我们可以将通过相同字符到达相同目的状态的源状态进行合并，以减少TCAM的表项数。

**表4.4 压缩之后的TCAM表**

|  |  |  |
| --- | --- | --- |
| **源状态** | **输入字符** | **目的状态** |
| \*\*\*\*\* | a | 01111 |
| 101\*\* | a | 10001 |
| 11111 | a | 10011 |
| 11\*\*\* | a | 10010 |
| 1001\* | a | 10000 |



**图4.4 DTSE算法流程图**

以字符a为例，表4.3是字符a相关的所有TCAM表项。但可以发现，状态节点0、2、8、12之间存在父子关系，状态节点0、2、8、12都通过字符a到达目的状态0，又因为状态节点2、8、12都是状态节点0的子节点，则可以将这四条迁移边合并成一条源状态编码为\*\*\*\*\*，字符为a，目的状态编码为00000的TCAM表项。

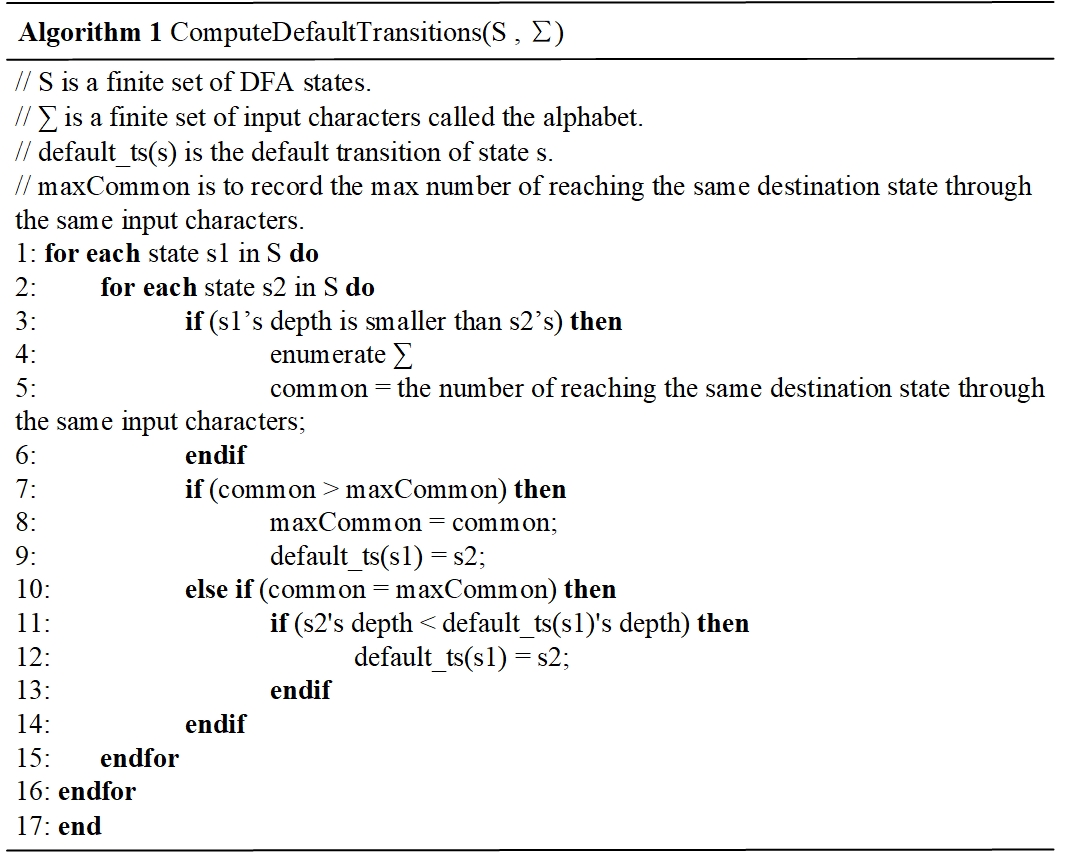
完成父子节点迁移边融合之后，我们还会对TCAM表项之间的只有一位不同源状态编码通过相同字符到达相同目的状态，我们将这类迁移边进行融合，成为兄弟节点迁移边融合。在这个例子中，不用进行兄弟节点迁移边融合。

最终压缩后的TCAM表项如表4.4。

4.2 算法实现

算法流程图如图4.4。

计算默认迁移边的伪代码如下，3-6行计算默认迁移边，7-14行与所记录的最大默认迁移边比较，找出最大的深度最低的默认迁移边。



4.3 算法复杂度分析

算法包含4个步骤，总体的时间复杂度为O(n2)，具体分析如下。

首先进行状态编码，包括构建默认迁移树、计算每个状态的编码宽度、对源状态目的状态编码这三个步骤，时间复杂度主要是由构建默认迁移树决定，因其使用的是D2FA默认迁移边的思想，其时间复杂度为O(n2)，所以状态编码的时间复杂度为O(n2)。最后进行迁移边融合，时间复杂度为O(n2)。

4.4 小结

本章提出了一个基于默认迁移边的状态编码算法，从算法思想、算法实现、算法复杂度三个角度进行阐述，DTSE通过构建默认迁移树对DFA中的状态节点进行源状态编码和目的状态编码，结果显示DTSE能够有效减少TCAM表项的数目，降低TCAM能耗、增加TCAM匹配吞吐量。

第五章 实验评估

5.1 实验环境

本文实验采用Intel Xeon Silver 4116 CPU @ 2.10GHz服务器，所使用的配置：操作系统CentOS 7 x86\_64，主存256GB， 12核心24个线程的CPU，L3缓存16.5MB，L2缓存1MB，L1缓存32KB。

模拟实验采用Snort和Suricata两种规则集，具体信息如下表5.1和表5.2。Snort和Suricata规则集的形式如：

alert tcp $HOME\_NET any ->

any 3689 ( msg:"APP-DETECT Apple iTunes client request for server info";

flow:to\_server,established; content:"cabab"; flowbits:set,itunes.serverinfo.request;

flowbits:noalert; service:http; reference:url,www.apple.com/itunes/;

classtype:misc-activity; sid:13898; rev:8; )

从这条规则我们可以解析出字符串cabab，然后检查该字符串是否出现过，若没出现过，则加入字符串集合。在解析规则集时，自动跳过被注释掉的规则和空行。

**表5.1 Snort规则集信息表**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **索引** | **规则总数** | **包含的规则集文件** | **规则数** | **状态数** |
| SN1 | 747 | snort3-malware-other.rules | 301 | 26476 |
| snort3-file-office.rules | 446 |
| SN2 | 700 | snort3-server-webapp.rules | 234 | 17946 |
| snort3-file-pdf.rules | 466 |
| SN3 | 1221 | snort3-protocol-snmp.rules | 1 | 17595 |
| snort3-browser-ie.rules | 1220 |
| SN4 | 486 | snort3-protocol-imap.rules | 2 | 13330 |
| snort3-protocol-voip.rules | 2 |
| snort3-sql.rules | 16 |
| snort3-file-pdf.rules | 466 |
| SN5 | 747 | snort3-browser-plugins.rules | 31 | 20856 |
| snort3-os-other.rules | 38 |
| snort3-malware-backdoor.rules | 110 |
| snort3-server-other.rules | 212 |
| snort3-server-webapp.rules | 234 |
| snort3-file-other.rules | 269 |

**表5.2 Suricata规则集信息表**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **索引** | **规则集名** | **包含的规则集文件** | **规则数** | **状态数** |
| SU1 | 4725 | emerging-inappropriate.rules | 1 | 25270 |
| emerging-web\_specific\_apps.rules | 4724 |
| SU2 | 1041 | emerging-info.rules | 491 | 13560 |
| emerging-policy.rules | 550 |
| SU3 | 934 | emerging-attack\_response.rules | 68 | 16227 |
| emerging-chat.rules | 77 |
| emerging-web\_client.rules | 239 |
| emerging-policy.rules | 550 |
| SU4 | 774 | emerging-mobile\_malware.rules | 179 | 17545 |
| emerging-activex.rules | 186 |
| emerging-sql.rules | 191 |
| emerging-scan.rules | 218 |
| SU5 | 453 | emerging-worm.rules | 11 | 10505 |
| emerging-smtp.rules | 13 |
| emerging-tftp.rules | 13 |
| emerging-icmp\_info.rules | 14 |
| emerging-scada.rules | 16 |
| emerging-exploit.rules | 386 |

关于本次毕业设计的代码量，一共实现了实际DFA、CSTSE、CSE、DTSE四个算法，其中实际DFA、CSTSE、CSE是已有的算法，DTSE是提出的新算法。总的代码行数有7300行，每个算法的代码行数统计如下表5.3。

**表5.3 代码行数统计表**

|  |  |
| --- | --- |
| **算法** | **代码行数** |
| 实际DFA | 1550 |
| CSTSE | 1924 |
| CSE | 1884 |
| DTSE | 1942 |

5.2 实验结果

5.2.1 TCAM表项数

图5.1和图5.2分别是在Snort规则集、Suricata规则集下的TCAM表项数的模拟结果。与理论DFA和实际DFA相比，CSTSE、CSE、DTSE三种算法都能很好地压缩TCAM表项数，CSTSE和DTSE的压缩效果最好，都能压缩到最佳效果，压缩后的TCAM表项数是理论DFA的0.4%左右，且与状态数相等。而CSE的TCAM表项数大约是是CSTSE、DTSE的TCAM表项数的2-3倍，压缩后的TCAM表项数约是理论DFA的1%左右。

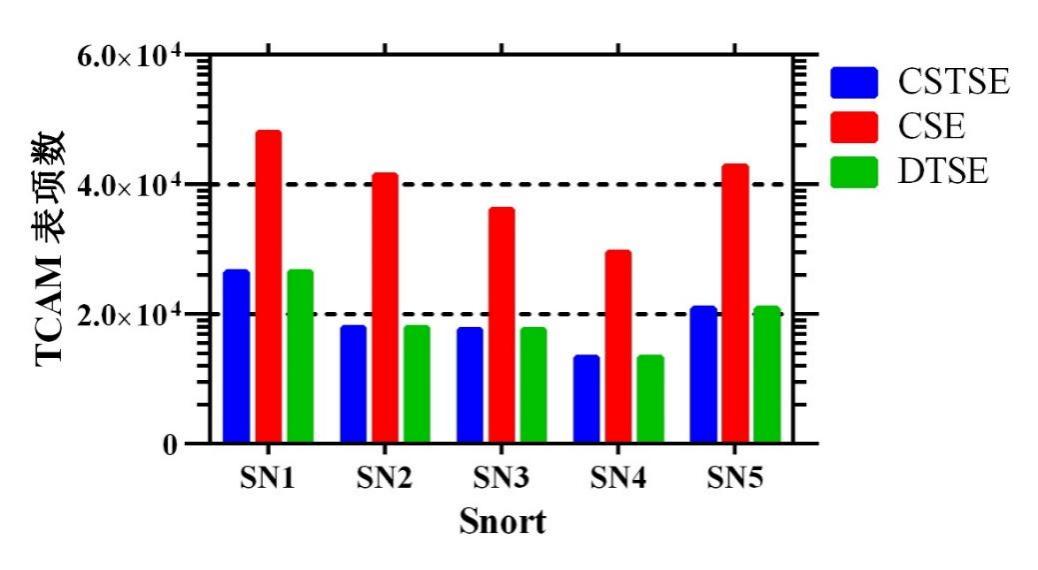
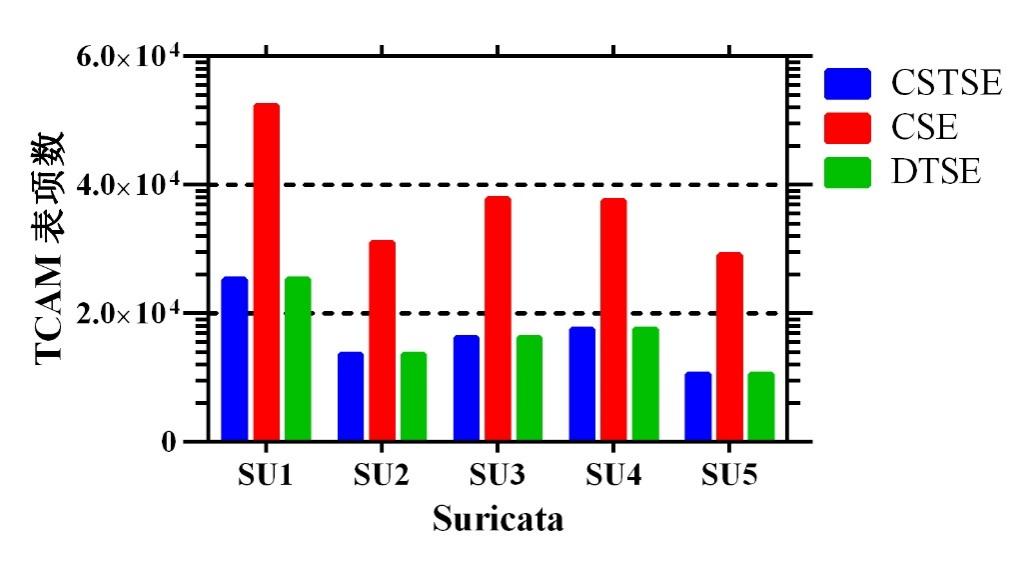
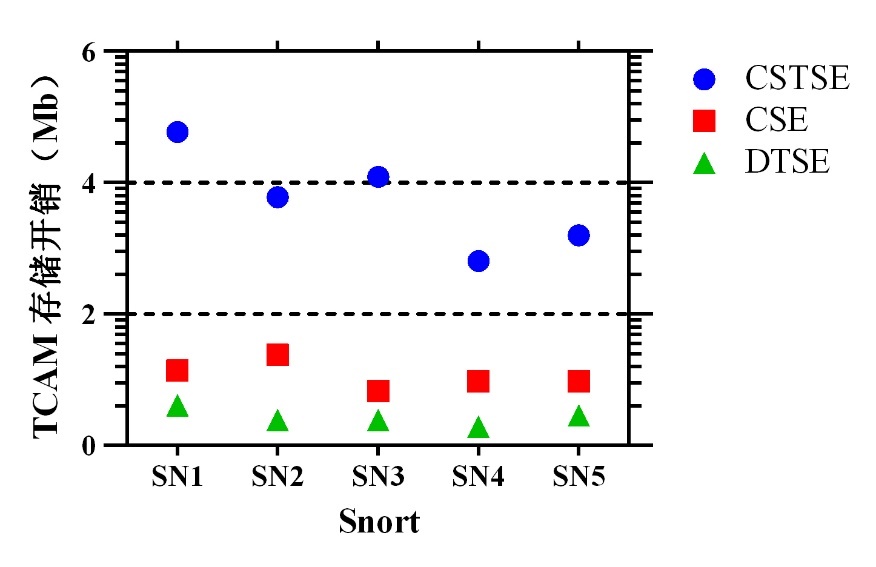


图5.1 Snort规则集下的TCAM表项数比较



**图5.2 Suricata规则集下的TCAM表项数比较**

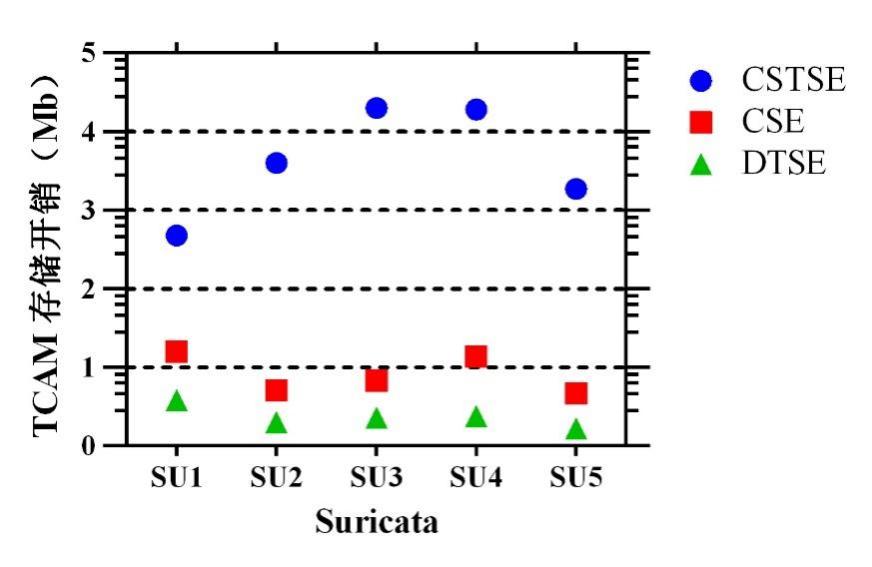
5.2.2 TCAM表存储开销



**图5.3 Snort规则集下的存储开销比较**

如图5.3、图5.4所示，DTSE存储开销是最小的。CSTSE的存储开销大约是理论DFA的3%左右，CSE的存储开销大约是理论DFA的1%，而DTSE的存储开销是理论DFA的0.4%。

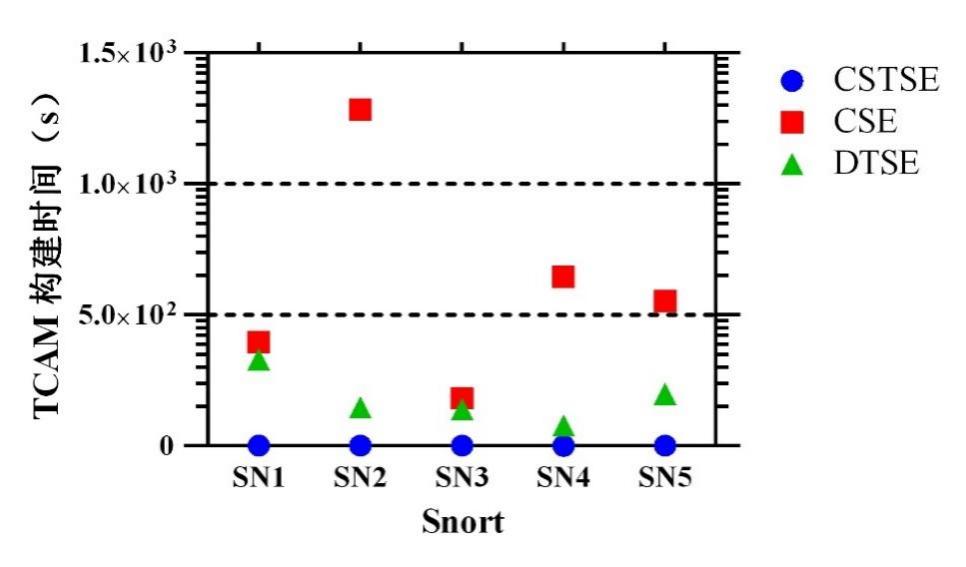
存储开销由迁移边数和编码宽度两个因素同时决定，CSE的TCAM表项数是DTSE的2-3倍，所以CSE的存储开销始终比DTSE开销大。而CSTSE的编码宽度由DFA树中最长的路径决定，CSTSE的编码宽度过宽，所以CSTSE的存储开销比DTSE多了一个数量级。同时，这也表明，CSTSE的致命缺点就是它的编码宽度，因编码宽度过宽导致的空间消耗大。



**图5.4 Suricata规则集下的存储开销比较**

5.2.3 TCAM表构建时间

以Snort规则集对TCAM构建时间为例进行分析。

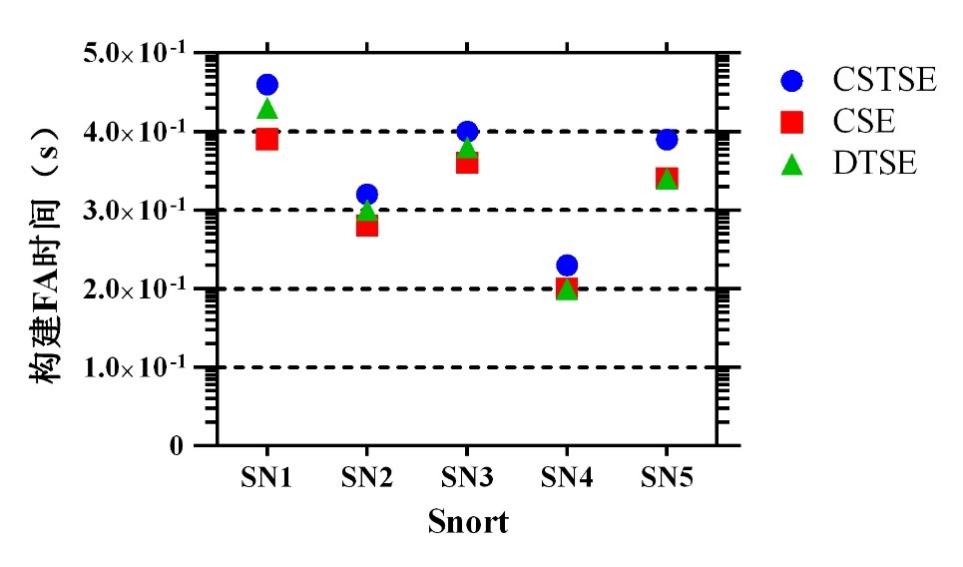


**图5.5 总时间比较**

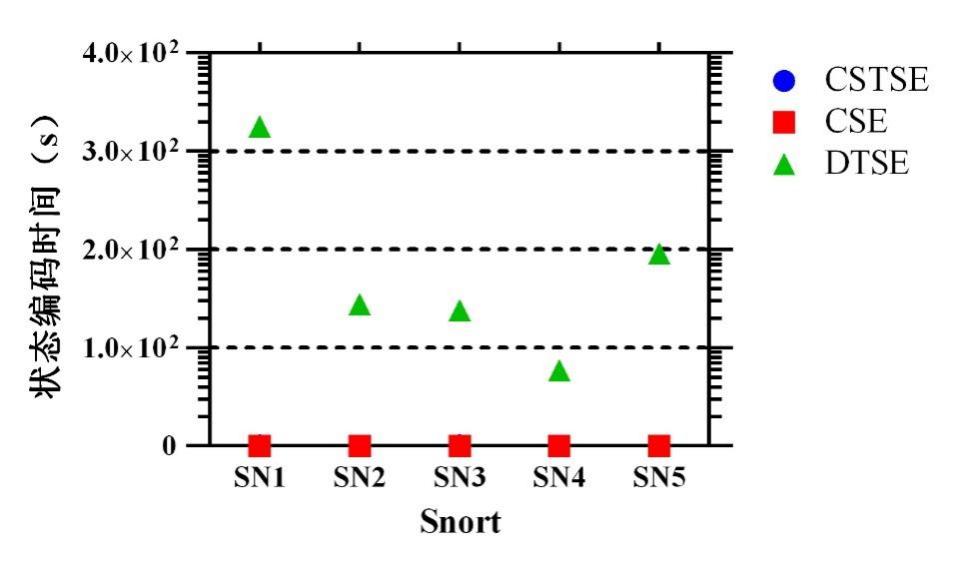
图5.5是Snort规则集下的总时间的比较，从图中可看出CSTSE所用时间最小。总的构建时间包括3个步骤：构建FA、状态编码、迁移边融合。

图5.6是在Snort规则集下构建FA的时间比较，由图可知，三个算法构建FA的时间差距特别小，在同一个数量级别。

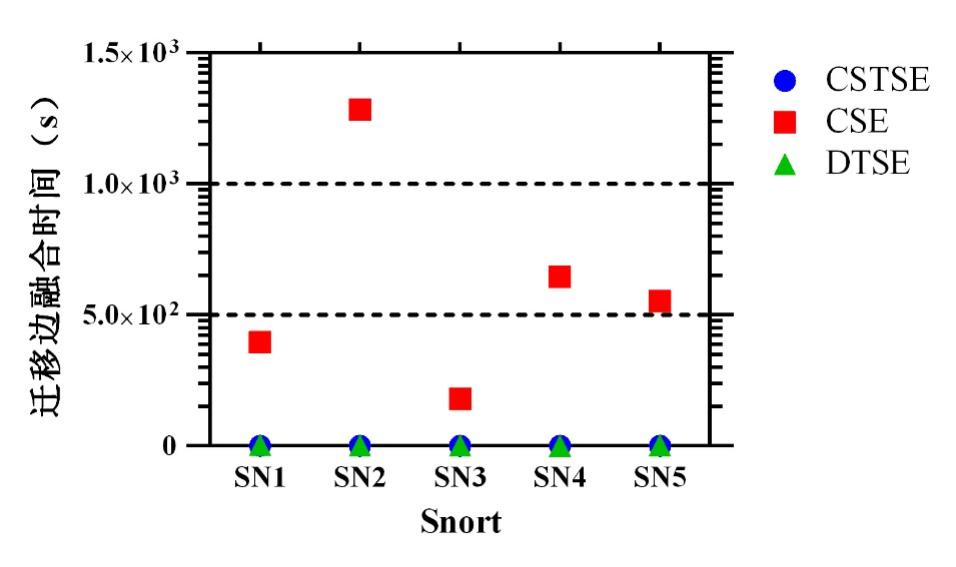
图5.7是CSTSE、CSE、DTSE在进行状态编码的时间。从表中可以看出，DTSE所耗时间最多，因为其构造默认迁移树的时间复杂度为O(n2)。而其他两个算法的时间复杂度为O(nlogn)。CSTSE比CSE算法所耗时间多的原因是CSTSE编码过程比较复杂。



**图5.6 构建FA的时间比较**



**图5.7 状态编码的时间比较**

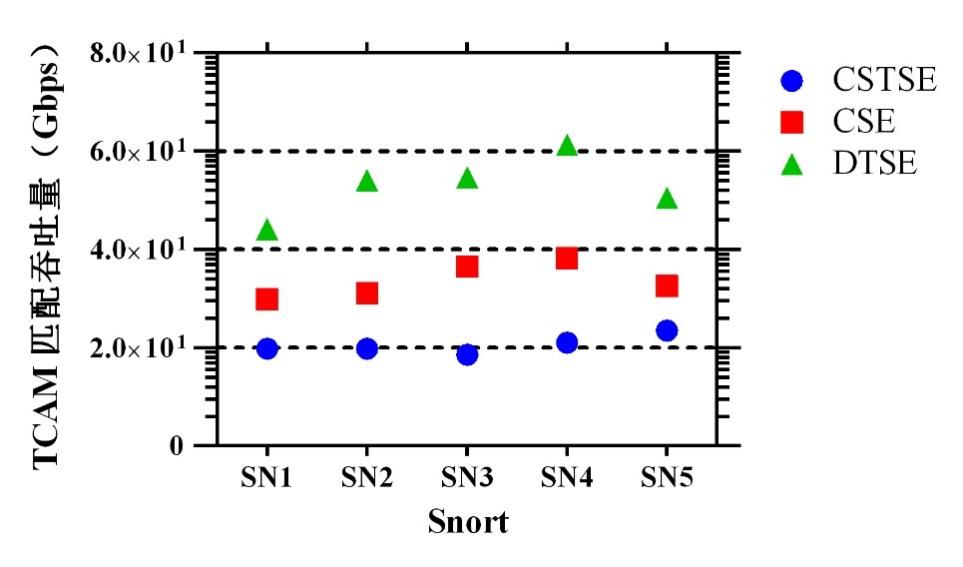


**图5.8 迁移边融合时间比较**

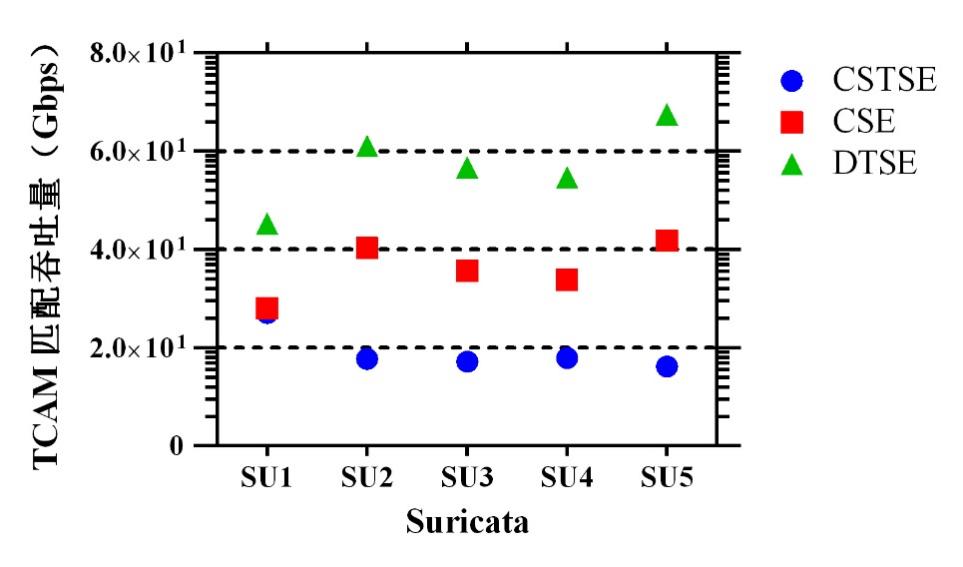
图5.8是CSTSE、CSE、DTSE在进行迁移边融合所需要的时间。CSTSE与DTSE融合时间相近，但仍可看出DTSE比CSTSE所耗费的时间多一点点。CSE和DTSE的融合逻辑一样，但因CSE融合次数比DTSE多，所以融合时间比DTSE长。

衡量TCAM的指标有两个：吞吐量(单位为Mb)和能耗(单位为nJ)。本文实验中TCAM芯片采用的28ns工艺、分区数为128。接下来从TCAM匹配吞吐量和能耗对算法进行分析。

5.2.4 TCAM匹配吞吐量



**图5.9 Snort规则集下的TCAM吞吐量**

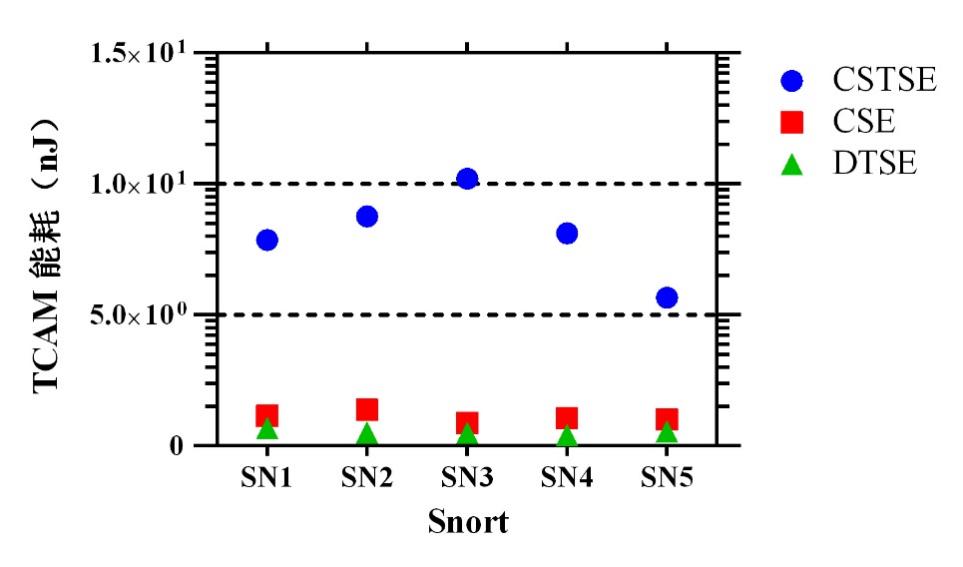


**图5.10 Suricata规则集下的TCAM吞吐量**

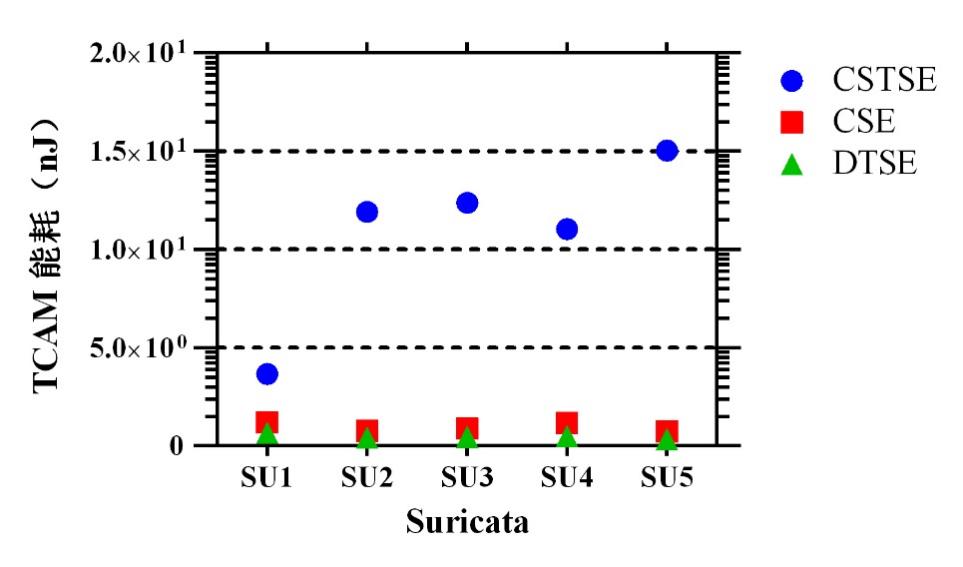
从图5.9、5.10可见，DTSE的吞吐量最大，CSE的吞吐量在DTSE与CSTSE之间，CSTSE算法的编码宽度过宽也导致其吞吐量不高。CSTSE、CSE的吞吐量是理论DFA的一千来倍，而DTSE可高达两千多倍。在Suricata的SU1这个规则集中，CSE和CSTSE差不多。在Snort和Suricata这10个规则集中，DTSE的匹配吞吐量都在40Gbps以上，CSTSE、CSE这两个算法的匹配吞吐量几乎都低于40Gbps，由此可看出，DTSE算法在TCAM匹配吞吐量方面的优势很明显。

5.2.5 TCAM能耗

从图5.11、5.12可知，DTSE的能耗最小，都不超过1 nJ，CSE仅次于DTSE，在Suricata规则集中，CSE和CSTSE的能耗都特别小，而CSTSE的TCAM能耗较高，且不太稳定，波动比较大。CSE比DTSE的能耗高约1个数量级，而CSTSE的TCAM能耗比DTSE的TCAM能耗高达2个数量级。



**图5.11 Snort规则集下的TCAM能耗**



**图5.12 Suricata规则集下的TCAM能耗**

从图5.11、5.12可知，DTSE的能耗最小，都不超过1 nJ，CSE仅次于DTSE，在Suricata规则集中，CSE和CSTSE的能耗都特别小，而CSTSE的TCAM能耗较高，且不太稳定，波动比较大。CSE比DTSE的能耗高约1个数量级，而CSTSE的TCAM能耗比DTSE的TCAM能耗高达2个数量级。

5.3 小结

本章对CSTSE、CSE、DTSE三个算法的实验环境、以及在Snort规则集和Suricata规则集上就TCAM表项数、存储开销、构建时间、TCAM的匹配吞吐量和能耗这几个方面进行比较。

就最终TCAM表项数而言，DTSE和CSTSE的最终压缩的表项数一样，CSE的TCAM表项数大概是CSTSE和DTSE的2-3倍。

在存储空间上，因为CSTSE的编码宽度由DFA树中最长的路径决定，对于大规模数据集，它的编码宽度就会远大于DTSE和CSTSE的编码宽度，又因为CSTSE和DTSE最终的TCAM表项数相差不大，则CSTSE的所占空间比DTSE算法大一个数量级。

对于构建时间，DTSE和CSE的总体时间复杂度一样，但是在实际模拟实验中，DTSE所耗时间小于CSE，三个算法中，CSTSE的时间复杂度最低，在实际模拟实验中，运行时间也特别少。本文对于构建时间的差异是可容忍的。

对于TCAM的匹配吞吐量和能耗，这两个指标与TCAM表项数和宽度密切相关，DTSE表现最佳，其匹配吞吐量最大，能耗也最小。

因此，DTSE在压缩TCAM表项数和所占空间上表现优异，能够有效降低TCAM的能耗，提高TCAM的吞吐量。

第六章 总结与展望

本文进行了基于TCAM的字符串匹配压缩算法的实现与评估，实现了现有的基于公共后缀迁移边的状态编码算法和覆盖状态编码算法，提出了基于默认迁移边的状态编码算法。DTSE利用D2FA算法思想计算出每个状态的默认迁移边，以默认迁移边作为父节点来构建一颗默认迁移树，然后在默认迁移树的基础上进行源状态编码和目的状态编码，将通过相同字符到达相同目的状态的多条TCAM表项压缩成一条TCAM表项，这能够减少TCAM表项数，降低TCAM表存储开销。DTSE构建的默认迁移树较其他相类似的压缩算法而言是最优的，编码宽度和压缩后的TCAM表项数都达到最佳的结果。

本文使用Snort和Suricata两类规则集进行实验评估，每类规则集选出5个进行模拟实验。实现了CSTSE、CSE、DTSE、实际DFA输出四个算法，共有7300行代码。接着从TCAM表项数、TCAM表存储开销、TCAM表构建时间、TCAM匹配吞吐量和能耗等方面对CSTSE、CSE、DTSE算法进行分析，实验结果表明，DTSE在TCAM表存储开销上比CSTSE算法低一个数量级。因TCAM的价格昂贵、存储空间受限，在通用交换机中的存储空间为2K-20K，所以TCAM表项数这个指标非常重要。CSE压缩之后TCAM表项数是DTSE压缩之后的TCAM表项数的2-3倍。虽然CSTSE压缩之后的TCAM表项数和DTSE压缩之后的TCAM表项数一样，但CSTSE算法的编码宽度过宽，导致TCAM的存储开销过大、匹配吞吐量较低且能耗过高。总而言之，DTSE较CSE和CSTSE两个算法而言，DTSE在TCAM表存储开销、TCAM匹配吞吐量和能耗方面有明显的优势。

然而，本文目前的研究还远远不够，要想实现高速可扩展的字符串匹配压缩算法这一终极目标，仍有很多问题需要去解决。本文目前存在的不足：基于默认迁移边的状态编码算法因采用D2FA思想，计算默认迁移边的时间复杂度为O(n2)，算法时间复杂度较高，且DTSE算法的存储开销会呈指数增长，系统容易出现内存溢出错误或者频繁内存调度，这难以适应大规模的字符串集，希望之后能提出一个在状态编码上的时间复杂度低且压缩之后的TCAM表项数能达到最佳的高速可扩展的字符串匹配压缩算法。

**参 考 文 献**

1. Srinivasan V, Suri S, Varghese G. Packet classification using tuple space search[C]//Proceedings of the conference on Applications, technologies, architectures, and protocols for computer communication. 1999: 135-146.
2. Yingchareonthawornchai S, Daly J, Liu A X, et al. A sorted partitioning approach to high-speed and fast-update OpenFlow classification[C]//2016 IEEE 24th International Conference on Network Protocols (ICNP). IEEE, 2016: 1-10.
3. Daly J, Bruschi V, Linguaglossa L, et al. Tuplemerge: Fast software packet processing for online packet classification[J]. IEEE/ACM transactions on networking, 2019, 27(4): 1417-1431.
4. Zhang C, Xie G. MultilayerTuple: A General, Scalable and High-performance Packet Classification Algorithm for Software Defined Network System[C]//2021 IFIP Networking Conference (IFIP Networking). IEEE, 2021: 1-9.
5. Aho A V, Corasick M J. Efficient string matching: an aid to bibliographic search[J]. Communications of the ACM, 1975, 18(6): 333-340.
6. [Sailesh Kumar](https://dblp.uni-trier.de/pid/k/SaileshKumar.html), [Sarang Dharmapurikar](https://dblp.uni-trier.de/pid/06/802.html), [Fang Yu](https://dblp.uni-trier.de/pid/44/3505-2.html), [Patrick Crowley](https://dblp.uni-trier.de/pid/92/6939.html), [Jonathan S. Turner](https://dblp.uni-trier.de/pid/t/JonathanSTurner.html):Algorithms to accelerate multiple regular expressions matching for deep packet inspection. [SIGCOMM 2006](https://dblp.uni-trier.de/db/conf/sigcomm/sigcomm2006.html#KumarDYCT06): 339-350.
7. Becchi M, Crowley P. An improved algorithm to accelerate regular expression evaluation[C]//Proceedings of the 3rd ACM/IEEE Symposium on Architecture for networking and communications systems. 2007: 145-154.
8. Bremler-Barr A, Hay D, Koral Y. CompactDFA: Generic state machine compression for scalable pattern matching[C]//2010 Proceedings IEEE INFOCOM. IEEE, 2010: 1-9.
9. Yun S K. An efficient TCAM-based implementation of multipattern matching using covered state encoding[J]. IEEE Transactions on Computers, 2010, 61(2): 213-221.
10. Hopcroft J E、Introduction to Automata Theory[J]. Languages and Computations, 1979, 127.
11. Huang K, Ding L, Xie G, et al. Scalable TCAM-based regular expression matching with compressed finite automata[C]//Architectures for Networking and Communications Systems. IEEE, 2013: 83-93.
12. 黄昆, 张大方, 谢高岗, 等. 一种面向深度数据包检测的紧凑型正则表达式匹配算法[J]. 中国科学: 信息科学, 2010, 40(2): 356r370.
13. 彭坤杨. 基于 TCAM 的高速可扩展的正则表达式匹配技术[D]. 合肥: 中国科学技术大学, 2013.

**致 谢**

论文反复改改写写，致谢却不知如何下笔，大学四年收获颇丰，要感谢的太多，却又不知从何说起。

首先，我要感谢我的导师唐兵老师和我未来三年的研究生导师黄昆老师。唐老师算是我的科研“引路人”，因为有唐老师的帮助，我才得以在本科发表一篇SCI三区论文。是唐老师带我走向保研路，刚开始我对保研毫无了解，是唐老师让我知道我自己努把力也能保上研。每当在竞赛/学习上遇到问题，唐老师总是不厌其烦，为我指点迷津。推免的过程并不简单，整整三年的累积才有机会获得保研名额，然而获得保研名额也只是开始，面试择校这一过程也非常艰难。在夏令营的时候，我获得了中南大学和南方科技大学的offer，因为身边的同学对南方科技大学不太了解，而中南大学是个老牌985，当时我非常纠结，不知道选哪个学校，唐老师有帮我进行分析、给与我建议，“你的选择没有错” 这句话也让我更加坚定。唐老师知道我们本科生的课程学习占据太多时间没有时间了解其他知识，为了让我们增加知识面，唐老师课题组开会都会让我们积极参与进来。但说来惭愧，在大二大三看论文写论文期间，我总是拖延。真的非常感谢唐老师，千言万语也表达不了我的感激之情。

黄昆老师算是我科研路的“明灯”。黄老师帮我确定好了我的研究方向，他还说我将要走的路他已经走过一遍了，这就避免了在研究方向上踩雷的风险；黄老师真的特别认真负责，每次汇报研究进展他都是对我进行一对一指导。在跟着黄老师做事的这八个月时间里，我感觉自己真的成长了很多，黄老师每次都会及时指出我有做的不对的地方。虽然刚开始作为小白什么都不懂的时候倍受打击，但是后来慢慢地适应了黄老师的方式。以前阅读论文总是拖拖拉拉还读不到重点，是黄老师的每一次反问让我慢慢改变我的思考思维方式，还记得跟着老师读的第一篇论文我被打回了三四次，差不多花了一个多月才稍微读懂那一篇论文，但现在我一周不到的时间就可以读懂一篇论文，明显有了很大的进步。当然，黄老师不只是否定我，在我有进步的时候他也会对我进行表扬。我也非常清楚地知道老师对我的否定是为了让我提升自己，虽然自己现在还有很多做的不够好的地方，在今后的时间里，我会尽量克服自己的缺点，努力提升自己，尽早达到老师的标准。

我还要感谢我的系主任姜磊老师、班主任陈燕晖老师。姜老师为人真的特别和善，是姜老师带着我们做课设，然后让我们了解前后端的一些主流框架并进行实战；在跟着姜老师项目小组做项目的过程中，我积累了一些Web项目的实战经验，也锻炼了我自主解决问题的能力；在老师的支持下，我也以第一作者身份申请了两项软件著作权。陈老师作为学院集训队的指导老师，他严谨认真的态度深深感染了我，当我遇到问题求助陈老师的时候，他会非常认真地全方位对问题进行分析，有很大的参考价值。感谢所有给予过我帮助的老师。

我要感谢已经毕业的17级学姐唐璐婕，每当我遇到问题或者烦恼，唐璐婕学姐都会耐心回答并给与支持，她在保研这条路上给了我很大的帮助。感谢相处了四年、无论严寒酷暑都一起去自习室学习、相互督促、共同成长的彭瑾璇同学、许郡怡同学和贺茜同学，大学四年也因为有你们而更加精彩。感谢计算机学院有过交集的每一位同学。

我要感谢我的父母，为了让我接受完整的教育，他们在外辛苦劳作毫无怨言。每当想到父母在外干苦力活，而自己却在享受着衣来伸手饭来张口的生活，我就更加坚定自己要努力读书的决心，想让他们以后也过上好生活。感谢我的姐姐，她从不会让我因为考试报名费昂贵而放弃，无论是精神上还是物质上她都会无条件支持我，是她一直鞭策着我鼓舞着我。

回想刚刚步入科大的我，性格内向。但在这四年里，从学习、科研、竞赛、考证到学生工作，每一种生活的可能性我都努力去尝试，一步一个脚印做到更好。未来我也会继续积蓄力量，砥砺前行！