入侵检测系统中的多模式精确匹配算法 WDawgMatch[1]

宁卓 龚俭

（东南大学计算机科学与工程学院 江苏南京 210096）

（江苏省计算机网络技术重点实验室）

摘要 经典的多模式匹配算法如 AC、BM，并不满足 NIDS 对报文负载中攻击特征串检测时做在线乱序流匹配的需求。著名的多模式精确匹配算法 DawgMatch 弥补了上述算法无法在扫描的同时获得分片的摘要信息的缺点，因此在网络入侵检测系统（NIDS）在线检测中得到普遍应用。尽管基于 DAWA 自动机使得 DawgMatch 可通过二元索引来提高空间使用效率，但它的匹配性能尚不能达到高速报文入侵检测线速匹配的要求。本文提出了新算法 WDawgMatch，它牺牲预处理时间，引入加权边消除了 DawgMatch 匹配回溯现象，提升匹配速度。性能分析和实验结果表明 WDawgMatch 降低了原算法的最坏时间复杂度，缩小了与 AC 算法的差距，完全满足 NIDS 线速匹配的要求。

关键词 DAWA DawgMatch 匹配回溯

An Accurate Multi-pattern Matching Algorithm WDawgMatch In Intrusion Detection System

Ning Zhuo Gong Jian

(School of Computer Science and Technology, Southeast University, Nanjing 210096)

Abstract The traditional multi-pattern matching algorithms like AC, BM do not meet the requirement of online out-of-order stream reassembly when NIDS detects attack signature matches within packet payloads. As a famous accurate multi-pattern matching algorithm, DawgMatch was generally used in NIDS as it could get the digests of the segment being scanned. Unfortunately, though it promotes the space usage by 2-tuple indexing factor with the help of DAWA automaton, its matching speed still can not catch up with the need of online linear detecting. To promote the performance of DawgMathch, we design a new algorithm WDawgMach based on it. WDawgMach makes use of weighted edges to eliminate back trace problem of DawgMatch to achieve linear matching speed. The performance analysis and experience shows that, by sacrificing the preprocessing time, WDawgMach improved the worst time complexity of DawgMatch and make it comparable to the algorithm AC.

Keywords DAWA, online out-of-order reassembly, packet scan

1 引言

多模式精确匹配是指在一次扫描内完成对若干个特征串的查找。该技术广泛用于文献查找、安全检测、基因串匹配等领域。经典的算法包括 KMP（Knuth-Morris-Pratt）、AC（Aho Corasick）、BM（Boyer Moore）等[1][2][3]，这些算法匹配速度与输入正文长度成正比，与特征串长度和个数无关，匹配性能优越。但是在 NIDS 对乱序流的攻击特征串进行在线检测时，上述经典算法都不适用。这是因为在线检测不可能象离线检测那样采用缓存机制，它要求在扫描分片的同时进行分片拼接，因而要求匹配算法具有记录分片摘要信息的能力。显然以上经典算法都不能到达上述要求。DawgMatch[4]

[1]本文得到国家重点基础研究发展规划973计划（2009CB320505）；国家科技支撑计划（2008BAH37B04）资助。

作者简介：**宁卓**（1975－），女，广西省玉林市人，博士生，主要研究方向为网络安全检测，Email：zhning@njnet.edu.cn。电话：025-83794000-401。**龚俭**（1957－），男，上海市人，教授，博士生导师，主要研究方向为网络安全，网络管理和网络体系结构等。电话：025- 83794341

是由 M.Crochemore 通过引入回溯边将 DAWA 进行扩展设计出的一个多模式精确匹配算法，具有记录分片摘要信息的能力，但是其算法复杂度与模式串的长度成正比，因此离在线检测的线速匹配要求还有一定差距。本文通过分析 DawgMatch 的弱点，引入加权边消除了 DawgMatch 匹配回溯现象，设计出了一个新算法 WDawgMatch。通过牺牲预处理时间的策略，将 WDawgMatch 的匹配速度提升至线速，充分满足了 NIDS 在线乱序流检测的需要。

本文第二节简单介绍了 DawgMatch 算法，分析了导致其匹配速度慢的原因，有针对性地提出引入加权边消除匹配回溯现象的方法，并形式化地证明了方法的正确性。第三节提出了 WDawgMatch 算法，包括生成和匹配算法，并对算法性能进行了时空复杂度分析。第四节描述了实验结果。

2 算法改进

2.1 DawgMatch 算法

在描述算法之前，将相关定义说明如下：P 为模式串集合，A 表示文字集合，

定义 2-1（**最长模式因子后缀**定义）[5]：已知正文串 T，模式串集为 P，最长模式因子后缀= longest{s | $\forall s \in$ 字符集 A^*， $s \in$ factor(P) $\wedge s \in$ suff(T)}，记作 **lsuff**(T)。例如，T=”*This is Text*”, P = {“*u xt v*”},则 lsuff(T) = “*xt*”。

定义 2-2（**右等价代表因子**定义）[5]：已知字符串 s，其右等价集为$[s]_R$，$[s]_R$ 的右等价代表因子 = {l | 字符串 $l \in [s]_R \wedge \forall$ 字符串 $s' \in [s]_R, s' \in$ suff(l)}，记作 $\mathbf{imp_R}([x]_R)$。例如 P = {’*abcab*’}，[‘*b*’]$_R$ ={‘*b*’, ‘*ab*’}，imp$_R$ ([‘*b*’]$_R$) = ‘*ab*’ 。

DawgMatch 的基本思想是[4]：正文串 T 任意匹配子串都能被 T 的某个前缀的最长模式因子后缀包含。因而，DawgMatch 算法遍历 T 的前缀 S，并计算当前最长模式因子后缀 lsuff(S)，然后在 lsuff(S) 中查找 P 在 T 中的匹配子串，就可以计算出 match_set(T,P)。其具体算法如下：

输入：正文串 T，模式串 P，Dawg(P)；输出：match_set(T,P)；

算法：读头从左到右逐字符扫描 T，读头当前字符 a_{m+1}。

步骤 1：已遍历的 T 的前缀为 S，设 lsuff(S)为 s(用一个二元组<q, l>表示 s)。计算 lsuff（$S\,a_{m+1}$）。计算分为两种情况：

前进：若 $sa_{m+1} \in$ factor(P)，有 lsuff（Sa_{m+1}）=sa_{m+1}。DAWG(P)上存在从当前状态 q 到状态 p 的以 a_{m+1} 为标号的前进边，则 s 修改为 sa_{m+1}，即更新二元组为<p, l + 1 >，读头前进。

回溯：若 $sa_{m+1} \notin$ factor(P)，则 lsuff（Sa_{m+1}）$\neq sa_{m+1}$，设 lsuff（Sa_{m+1}）=$s'a_{m+1}$，易知 $s' \in$suff(s)。此时利用回溯边可以由长到短遍历 suff(s)。直到找到 s',使 $s'a_{m+1}$ = lsuff(sa_{m+1}), 设置 s 为 s', 返回到前进情况。

步骤 2：如果 lsuff(Sa_{m+1})包含模式串 p（通过 lsuff(Sa_{m+1})对应的 DAWG(P)节点是否为终止节点以及因子的长度来判断），添加[p,m]至 match_set(T,P)。

2.2 改进思路

从算法易知 DawgMatch 匹配时计算复杂度主要由步骤 1 决定。考察完成单字符扫描时所需的操作数：前进情况下，操作数为常数；而在回溯情况下，最坏情况下需要回溯遍历当前最长模式因子后缀 s 的所有的右等价集祖先集，其回溯长度小于|s|-1。设正文串的长度为|T|，模式集为 P，最长模

式串长为 m，DawgMatch 算法匹配最坏时间复杂度为 O($m|T|$)，正比于模式串长度。而这导致了 DawgMatch 不能满足要求复杂度为 O($|T|$)的在线检测需求。为了克服回溯造成的性能低下，如果产生回溯时能够提供某种机制直接跳转，就可以降低匹配复杂度。

下列推导证明了已知 lsuff($s'a$)，就可以根据 $s'a$ 与 lsuff($s'a$)的关系直接计算出 lsuff(sa)，并引入加权边以记录 lsuff($s'a$)，产生回溯时直接跳转到下一个匹配位置，从而实现了线速匹配。

定理 2-1：已知字符集 A，模式串集合为 P。若 lsuff(s)=s，$s\in A^*$，则$\forall s' \in$suff(s)，lsuff(s')=s'。

证明：

$\mathbf{Q}\ lsuff(s) = s.$

$\therefore$ 根据定义2-1，$s \in factor(P)$

$\mathbf{Q}\ s' \in suff(s) \wedge s \in factor(P)$

$\therefore s' \in factor(P)$

$\mathbf{Q}\ s' \in suff(s') \wedge s' \in factor(P)$

$\therefore$ 根据定义2-1，$lsuff(s') = s'$

证毕。

推论 2-1：已知字符集 A，模式串集合 P。$\forall s\in A^*$，设 $imp([s]_R)_R$ 为 s'，若 lsuff($s'a$)=$s'a$，$a\in A$，则 lsuff(sa)=sa。从定理 2-1 直接可推出。

定理 2-2：已知字符集 A，模式串集合为 P。$\forall s\in A^*$，若 lsuff(s) $\neq s$，设 lsuff(s)=s'。则$\forall s''\in$suff(s)，lsuff(s'')=s'。

证明：

$\mathbf{Q}\ lsuff(s) = s'.$

$\therefore$ 根据定义2-1，$s'. \in suff(s)$ (1)

设 $lsuff(s'') = t$

$\therefore$ 根据定义2-1，$t \in suff(s'')$

$\mathbf{Q}\ s'' \in suff(s) \wedge t \in suff(s'')$

$\therefore t \in suff(s)$ (2)

根据t与s'的关系，否则分两种情况：

- 情况 1。t=s'，定理 2-2 得证。
- 情况 2。$s' \in$suff(t)$\wedge t \neq s'$。

  $\mathbf{Q}\ t = lsuff(s'')$

  $\therefore$ 根据定义2-1，$t \in suff(P)$

  $\therefore$ 根据(2)，$t \in suff(s) \wedge t \in suff(P)$

  $\therefore$ 根据定义2-1，$t \in suff(lsuff(s))$，即$t \in suff(s')$

  $\therefore$ 与情况1相悖

- 情况 3。$t\in$suff(s')$\wedge t \neq s'$。情况 3 也不存在，证明过程与情况 2 相同。

因此，lsuff(s'')=s'。

证毕。

推论 2-2：已知字符集 A，模式串集合 P。$\forall\ s\in A^*$，设 $imp([s]_R)_R$ 为 s'，若 lsuff($s'a$)$\neq s'a$，$a\in A$，则 lsuff(sa)= lsuff($s'a$)。

证明:

设 lsuff($s'a$) = t',$t'\in A^*$。

$\mathbf{Q}\ imp([s]_R)_R = s'$

$\therefore$ 根据定义$2-2$，$s \in suff(imp([s]_R)_R) = suff(s')$

$\therefore sa \in suff(s'a)$

$\mathbf{Q}\ lsuff(s'a) = t'$

$\therefore$ 根据定义$2-1$，$t' \in suff(s'a)$

根据t'与sa的关系，分为两种情况：

- 情况1，$sa \in$suff(t')。

  $\mathbf{Q}\ t' \in factor(P) \wedge sa \in suff(t')$

  $\therefore sa \in factor(P)$

  $\mathbf{Q}\ s' \in [s]_R \wedge sa \in factor(P)$

  $\therefore s'a$和sa在P上右等价，$s'a \in factor(P)$

  $\mathbf{Q}\ s'a \in suff(s'a)$

  $\therefore$ 根据定义$2-2$，$s'a = suff(s'a)$，与推论$2-2$题干假设相悖

  $\therefore$ 情况1不成立

- 情况2，$t'$$\in$suff($sa$)$\wedge t' \neq$ suff(sa)，根据定理2-2可知lsuff(sa)= lsuff($s'a$)。

证毕。

定理2-3: 已知字符集A，模式串集合P。字符串$s \in A^*$，lsuff(s)=s'，则$\forall\ a \in A$，lsuff(sa) $\in$suff($s'a$)。

证明：

设$lsuff(sa) = t$，字符串$t \in A^*$

$\therefore$ 根据定义$2-2$，$t. \in factor(P). \Lambda t \in suff(sa)$ (1)

$\mathbf{Q}\ lsuff(s) = s'$

$\therefore$ 根据定义$2-2$，$s' \in suff(s)$，即$s'a \in suff(sa)$

根据t与$s'a$的关系分两种情况讨论：

若$t \in suff(s'a)$，证明成立

否则，$s'a \in suff(t) \Lambda t \neq s'a$ (2)

设$t = s''a$，字符串$s'' \in A^*$，由(2)易知$s' \in suff(s'') \Lambda s' \neq s''$

$\mathbf{Q}$由(1)和$t = s''a$可知，$t \in factor(P) \Lambda s'' \in factor(t)$

$\therefore s'' \in factor(P)$

由(1)可知，$t \in suff(sa)$，(3)

$\mathbf{Q}\ s'' \in suff(s)$，(4)

$\therefore$(3)、（4）根据定义$2-2$可知，$s'' \in suff(lsuff(s))$，即$s'' \in suff(s')$

$\therefore$根据$t = s''a$可知，$t \in suff(s'a)$，与(2)相悖

证毕。

推论2-3: 已知字符集A，模式串集合P。设字符串s，s' $\in A^*$，若$s'$$\in [s]_R$，则$\forall$ 字符$a \in A$，lsuff($s'a$) $\in [$lsuff(sa)$]_R$。

证明：

设s''=imp($[s]_R$)$_R$，证明，分两种情况：

- 情况1. lsuff($s''a$)=$s''a$。

$\mathbf{Q}\ s \in [s]_R \wedge s' \in [s]_R$

$\therefore$ 根据推论2−1，$lsuff(s'a) = s'a \wedge lsuff(sa) = sa$ (1)

$\mathbf{Q}\ s \in [s"]_R \wedge s' \in [s"]_R \wedge s"a \in factor(P)$

$\therefore$ 根据性质2-4，$sa =_R s'a$ (2)

$\therefore$ 结合(1)、（2）可知，$lsuff(s'a) \in [lsuff(sa)]_R$

- 情况 2. lsuff(*s"a*)≠ *s"a*。

  根据推论 2-2 可知 lsuff(*s'a*)= lsuff(*sa*)，即 lsuff(*s'a*) ∈[lsuff(*sa*)]$_R$。

证毕。

3 WDawgMatch 算法

设 imp([*s*]$_R$)$_R$ 为 *s*'，从推论 2-1、2-2 可以看出，如果事先已知 lsuff(*s'a*)，就可以根据 *s'a* 与 lsuff(*s'a*) 的关系直接计算出 lsuff(*sa*)。WDAWG 就是用加权边来记录 lsuff(*s'a*)，从而辅助 WDawgMatch 算法加快计算 lsuff(*sa*)。以下是 WDAWG 定义：

定义 3-1（**WDAWG** 定义）：已知模式集 *P*, WDAWG(*P*)={*Q, i, T, WE* }，*Q* = {[*s*]$_R$ | ϖ 模式 $p_j \in P$,字符串 $s \in$ factor(p_j)}，每个右等价集都等价于 *Q* 中的一个节点。*i* = [*λ*]$_R$，表示初始结点。*T* = {[*s*] $_R$ | 字符串 *s*, ϖ 模式 $p_j \in P, p_j \in$ suff(*s*)}，表示终止结点集合。**加权边集** *WE* ={([*s*]$_R$, *a*, *k*, [lsuff(*sa*)]$_R$) | 字符串 *s*∈*A**, *a*∈*A*，*k* 是整数，表示该边的权值}。根据推论 2-3 可知，[*s*]$_R$ 中的任意字符串 *s*'，lsuff(*s'a*)属于同一右等价集，因而以[*s*] $_R$ 为起点，[lsuff(*sa*)] $_R$ 为终点的边只有一条。假设 imp([*s*]$_R$)$_R$ 为 *s"*，该边权值 *k* 的含义分为两种：*k*=1，表示 lsuff(*s"a*)=*s"a*；*k*<=0，lsuff(*s"a*)就为 *s"a* 长度为-*k* 的后缀。因此，利用加权边 *k* 就可以记录 lsuff(*s"a*)。

3.1 WDawgMatch 生成算法

WDAWG 可以根据 DAWG 来进行转换，生成 WDAWG 特有的加权边。具体转换算法如下：

输入：字符表 *A*，模式串 *P*，DAWG(*P*)

输出：WDAWG(*P*)

算法：

- 步骤 1。已知 DAWG(P)={*Q, i, T, E, F*}，复制 *i*，*T, Q* 到 WDAWG(*P*)中；
- 步骤 2。遍历节点集 *Q*，当前节点为 *v*，相应的 WDAWG（*P*）的节点 *w*， *v* 的右等价代表因子 imp$_R$(*v*)为 *s*，
  - 遍历字符表 *A*，当前字符为 *a*;
  - 计算 *lsuff(sa)=s'*，对应的 DAWG(*P*)上的二元组为(*q*, |*s'*|)，*q* 相应的 WDAWG（*P*）的节点 *p*，计算步骤与 DawgMatch 算法步骤 1 一致;
  - 如果 *s'*=*sa*，在节点 *w* 中添加边(*v*, *a*, 1, *q*)，否则添加边为(*v*, *a*, -|*s'*|，*q*)。
- 步骤 3。输出 WDAWG(P)。

3.2 WdawgMatch 匹配算法

WDawgMatch 利用 WDAWG 加权边，改进了 DawgMatch 步骤 1 的操作，具体的算法如下：

输入：字符表 A，正文串 T，模式串 P，WDawg(P)；

输出：match_set(T,P)；

算法：读头从左到右扫描 T，读头当前字符 a_{m+1}。

- 步骤 1　已遍历的 T 的前缀为 S，设 lsuff(S)为 s（用一个二元组<q, l>表示 s），找到 q 上标号为 a_{m+1} 的加权边(q, a_{m+1}, k, p)。计算 lsuff（S a_{m+1}）=lsuff(sa_{m+1})：
  - 如果 k=1，表示 lsuff(s a_{m+1})=s a_{m+1}，更新二元组为<p, l+1>；
  - 否则，lsuff(s a_{m+1})为 s a_{m+1} 长度为-k 的后缀，更新二元组为<p, -k>
- 步骤 2　如果 lsuff(Sa_{m+1})包含模式串 p（通过 lsuff(Sa_{m+1})对应的 DAWG(P)节点是否为终止节点以及因子的长度来判断），添加[p，m]至 match_set(T,P)。

3.3 WdawgMatch 性能分析

空间复杂度分析较 DawgMatch 算法，WDawgMatch 算法的辅助数据结构将前进边扩展为加权边。在使用邻接矩阵表示自动机的前提下，DAWG 每条边记录了终节点的编号，WDAWG 增添了边的权值。假设权值与编号的空间代价相同，则 WDAWG 节点中存储每条边的空间代价增加一倍，但是 WDAWG 节点数没有发生变化，因而改进后现有空间最坏复杂度仍为 O(||P|||A|)。

时间复杂度分析在预处理计算上，WDawgMatch 算法需要额外计算每条边的权值，根据 3.1 节生成算法，每条边的权值计算需要根据该边所在节点的回溯边进行遍历，因而，计算复杂度上限为 m，m 为模式的平均长度，[4]已证明 DAWG 的节点最大值为 2||P||-1，每节点的边数上限为|A|，则 WDawgMatch 算法增加的操作数上限为 O(|A| m ||P||)，高于原预处理的时间复杂度。在匹配计算时，根据 WDawgMatch 算法，步骤 1 只包括一次比较和更新二元组操作。因而操作复杂度为常数，其他步骤根据 3.1 可知复杂度也为常数。考虑正文串的长度|T|，匹配计算的时间复杂度是 O(|T|)，匹配复杂度等同于 AC 等经典离线多模匹配算法。

4 实验

实验主要是测试 WDawgMatch 算法的运行时处理性能和预处理性能。本节利用 Abyss Bink 统计算法处理中所运行的用户态有效指令，将 WDawgMatch 与经典的离线算法 AC、DawgMatch 算法相比较进行性能分析。对于 Abyss Bink 的这种量化策略，有两点需要说明：

（1）算法的实际运行速度不仅与运行 CPU 所执行的指令数相关，还受到该 CPU 的流水线深度、一二级 Cache 大小及淘汰策略、分支预测技术等因素影响。不同的 CPU 相关实现技术存在很大的差别，因此即使指令数相同、主频相同，实际运行时间仍可能存在很大差别。本节是因为 WDawgMatch 的理论分析只考虑算法的操作数，所以相应地也只统计指令数；

（2）不同指令的执行周期有所不同，因而理论上利用执行周期固定的微指令来统计具有更好的参考性。但是在 Abyss Bink 的实际使用过程中，发现微指令统计在不确定的情况下，微指令总数明显异常（微指令数是指令数的 1，000 倍）。为了防止测试软件的偏差，仍然根据指令数统计。而且已知正常情况下微指令数与指令数的比值在 1.36~1.41 之间。所以按指令数进行统计还是可以接受的。

根据的性能分析得知，WDawgMatch 算法性能主要与模式集大小和模式的长度相关。因此设计下列四组实验：

4-1 **模式集大小对 WDawgMatch 运行时处理速度的影响** 设置模式为长度 20 的随机字串，模式集大小分别设为 50，100，150。正文串集合包含 60，000 个长度为 400 字节的随机字符串，循环播放 100 次，统计 AC，WDawgMatch，DawgMatch 三种算法的有效指令数。实验结果显示模式集大

小的改变对三种算法运行性能的影响很小，WDawgMatch 算法有效地缩短了 DawgMatch 算法和 AC 算法的差距。

4-2 模式集大小对 WDawgMatch 预处理时处理速度的影响　设置同 4-1。统计三种算法预处理的有效指令数。实验结果显示随模式集增大三种算法预处理指令数都相应增加，但是 WDawgMatch 算法较之其他两种线型增长的算法更是显现出超线性增长的趋势。这似乎与 3.3.4 节的分析相抵触。进一步分析可知随着模式集增大，尽管模式串长不变，DAWA 的深度还是会随模式集增大而加深，相应地 WDawgMatch 多计算加权边的开销也增大，呈现出非线性增长的趋势。

4-3 模式长度对 WDawgMatch 运行时处理速度的影响　设置模式集大小为 100，模式串长度分别设为 10，20，30。正文串集合如上述生成，统计三种算法的有效指令数。实验结果显示模式长度的改变对三种算法运行性能的影响很小，而 WDawgMatch 在模式长度为 30 时甚至出现了指令数下降的情况。分析得知：模式长度较长时随机正文串匹重的概率相应减小，匹配字串数下降会直接导致输出等运行时操作数下降。因而 WDawgMatch 性能甚至有所提升。

4-4 模式长度对 WDawgMatch 预处理时处理速度的影响　设置同 4-1。统计三种算法预处理的有效指令数。实验结果显示随模式集增大三种算法预处理指令数都相应增加，从理论分析可知 WDawgMatch 的预处理指令数与模式串长度的平方成线性正比，可见实验数据基本吻合。

图 4-1 不同规模模式集下三种算法运行性能比较图

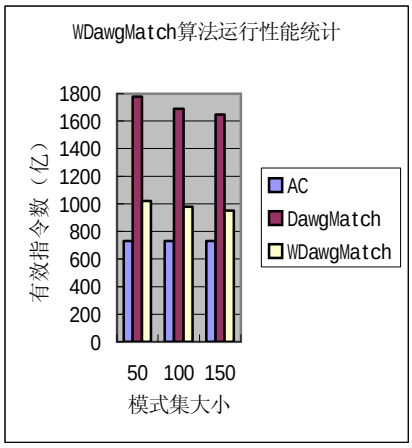


图 4-2 不同规模模式集下三种算法预处理性能比较图

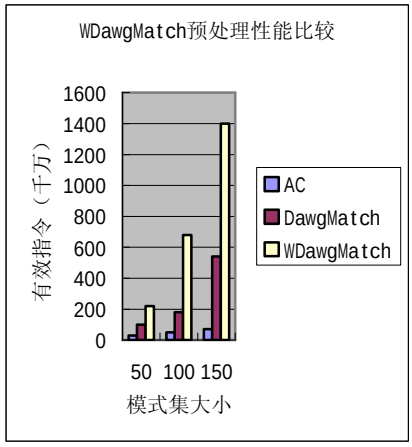


图 4-3 不同模式串长下三种算法运行性能比较图

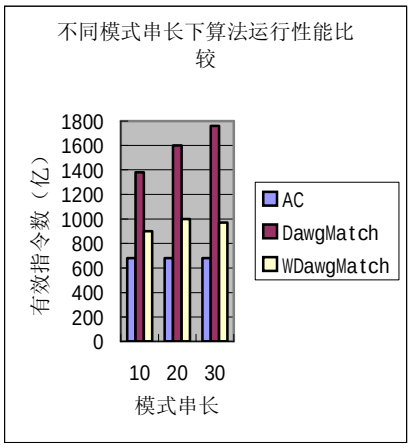
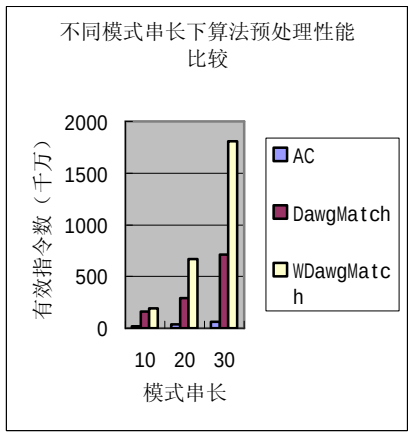


图 4-4 不同模式串长下三种算法预处理性能比较图



上述四组实验显示 WDawgMatch 有效缩短了 DawgMatch 和 AC 的性能差距，其算法性能与模式集大小和模式串长度无关。该算法的缺点是预处理时间过长，在 4-2、4-4 中可见预处理时间随模式集大小和模式串长度呈非线性增长，这一点和 3.3.3 中的复杂度分析 O(|A| m ||P||)一致。在 NIDS 应用背景下只是对算法的匹配速度要求高，由于 NIDS 的重启和规则库更新都不是很频繁，所以对

预处理效率不高导致规则集编译时间长的缺点是可以接受的。

5 结束语

本文针对 NIDS 在线乱序流特征串检测的特殊需求，利用加权边克服了 DawgMatch 的匹配回溯问题，将匹配复杂度从 O(*m*|*T*|)提升至 O(|*T*|)，满足了线性匹配要求。改进后的 WDawgMatch 算法匹配时间复杂度与模式集的大小和模式长度无关，只与输入正文串成线性正比。实验结果证明 WDawgMatch 的性能指标完全满足 NIDS 对在线乱序流匹配的要求。

参考文献

[1] A V. Aho, M. J. Corasick, Efficient String Matching: An Aid to Bibliographic Search, Communications of The ACM, Vol. 18, No. 6 Jun. 1975 , pp. 333-340,

[2] R.S. Boyer, J.S. Moore, A Fast String Searching Algorithm. Communications of the Association for Computing Machinery, Vol. 20, No. 10, 1977, pp. 762-772.

[3] M. Fish and G. Varghese, Fast Content-Based Packet Handling for Intrusion Detection, UCSD technical report CS2001-0670, 2001

[4] M.Crochemore, C. Hancart, Automata for matching patterns，in Handbook of Formal Languages, G. Rosenberg and A. Salomaa，eds., volume 2, Linear Modeling, Springer-Verlag, 1997. pp. 399-462

[5] A. Blumer, J. Blumer, A. Eherenfeucht, D. Haussler., R Mcconnell, Linear size finite automata for the set of all subwords of a word: An outline of results. Bull. Eur. Assoc. Theoret. Comput. Sci. no.21, 1983, pp 12-20.