基于字符频率及分治法的字符串模式匹配算法*)

邓一贵[1,2]

(重庆大学计算机学院 重庆 400044)[1] (重庆大学信息与网络管理中心 重庆 400044)[2]

摘 要 本文提出的基于字符使用频率及分治法的改进字符串模式匹配算法可以在扫描被匹配目标串时每次跳过的字符在统计结果上比目前广泛使用的 Boyer-Moore 算法跳过的字符更多,进一步减少了匹配的统计次数。

关键词 字符串模式匹配,字符使用频率,分治

String Pattern Matching Algorithm Based on Frequencies of Characters and Dividing and Conquering

DENG Yi-gui[1,2]

(College of Computer Science, Chongqing University, Chongqing 400044, China)[1]

(Information and Network Center, Chongqing University, Chongqing 400044, China)[2]

Abstract The skipped characters in the algorithm based on frequencies of characters and dividing and conquering are more in statistics than ones in Boyer-Moore algorithm popularly used at present. The matching statistical times using algorithm presented in the paper are reduced.

Keywords String pattern matching, Frequencies of characters, Divide and conquer

1 引言

根据入侵特征是否已知来分,入侵检测可以分为已知特征的误用检测和未知的异常检测。其中已知特征的误用检测往往使用大量的检测规则,规则中条件部分(特别是应用层协议的检测)通常要用到字符串模式匹配。因此,字符串模式匹配的效率好坏往往直接影响到入侵检测系统的已知特征的误用检测部分的性能。本文提出的基于字母使用频率及分治思想的改进字符串模式匹配算法可以在扫描被匹配目标串时每次跳过的字符在统计结果上比目前广泛使用的 Boyer-Moore 算法跳过的字符更多,进一步减少了整体匹配次数,提高了入侵检测系统误用检测中规则的特征匹配的速度。

2 基于字符串模式匹配的 BM 算法简介

字符串匹配是指在长度为 n 的字符串中检索长度为 m 的子串的所有出现。字符串匹配可分为单字符串匹配和字符串集合匹配两种。单字符串匹配算法最著名的是 KMP (Knuth-Morris-Pratt)算法和 BM(Boyer-Moore)算法。KMP 算法实现了无回溯匹配,字符串中的每个字符只匹配一次,时间复杂度为 $O(n+m)$[1];BM 算法采用跳跃方式,匹配时跳过不需匹配的字符,最优情况下的时间复杂度为 $O(n/m)$,平均情况下也大大优于 KMP 算法[2];文献[3]取掉 BM 算法的好后缀试探(Good Suffix Heuristic),形成 BMH(Boyer-Moore-Horspool)算法,实验证明 BMH 算法性能在自然语言环境下比原始 BM 算法还要好。

字符串集合匹配是从长度为 n 的字符串中一次查找 m 个字符串的所有出现,最经典的是 AC(Aho and Corasick)算法。该算法将待匹配的多个字符串转换为树状有限状态自动机,然后进行扫描匹配,最优情况和平均情况的时间复杂度都为 $O(n)$[4]。

Boyer-Moore 算法假设被匹配目标串为 T,长度为 n,模式串为 P,长度为 m,字符集 Σ 为数组{chr[0],chr[1],…,chr[$|\Sigma|-1$]},包含字符数量为 $|\Sigma|$,chr[i]表示字符集中第 i 个字符($0<i<|\Sigma|-1$)。

BM 算法的基本原理为:在 T 中间隔一定距离对一个字符进行匹配,根据当前被匹配的字符是否在 P 中出现及出现位置确定是否向右跳跃和跳跃距离。如果不跳跃,则在当前位置从右向左与 P 比较,否则根据预先计算好的跳跃距离跳到 T 中下一位置再比较。

BM 算法应用的关键在于预先根据模式串 P 构造针对每个字符的向右跳跃距离的 Shift 数组{Shift[hash(chr[0])], Shift[hash(chr[1]), …,Shift[hash(chr[$|\Sigma|-1$])]},这里的 Hash 函数把不同字符转换成 Shift 数组中的一个位置值 Hash(chr[i]),进而可以通过 Shift[hash(chr[i])]来得到字符 chr[i]失配时的跳跃距离。Shift 数组的计算方法如下:

如果字符 chr[i]在 P 中出现,Shift[hash(chr[i])]的值为 chr[i]在 P 出现的最右位置到模式串最右端的距离;如果字符 chr[i]在模式串 P 中不出现,Shift[hash(chr[i])]的值就为 m。

3 基于字符频率及分治法的字符串模式匹配算法

在明文字符表中,不同明文字符出现的频率往往不相同,如 26 个英文字符出现的频率,全部一、二级汉字的频率等。各种文字出现的频率是该种文字信息处理的基础,因此各国都有正确的统计,有的国家还列入标准化系列。表 1 是英文字母的出现频率[5],汉字的出现频率见文献[5]。

*)基金项目:重庆市自然科学基金项目(CSTC2007BB2178 和 CSTC2005BB2190)支持。邓一贵 博士研究生,主要研究方向为计算机网络及信息安全。

表 1　英文字母出现的频率表

英文字母	使用频率	英文字母	使用频率	英文字母	使用频率
A	0.0856	J	0.0013	S	0.0607
B	0.0139	K	0.0042	T	0.1045
C	0.0297	L	0.0339	U	0.0249
D	0.0378	M	0.0249	V	0.0092
E	0.1304	N	0.0707	W	0.0149
F	0.0289	O	0.0797	X	0.0017
G	0.0199	P	0.0199	Y	0.0199
H	0.0528	Q	0.0012	Z	0.0008
I	0.0627	R	0.0677		

如果在字符串模式匹配的坏字符移动时考虑使用字符的使用频率进行启发，则在匹配时会加大坏字符的失配率概率，从而提高移动的统计速度。

本文的 BM_in_frequence_match 算法引入了字符使用频率对字符匹配进行启发跳跃，尽可能选择失配概率高的字符进行比较，加大字符失配的概率，从总体上加快字符移动统计速度。具体算法如下：

```
函数 BM_in_frequence_match(string T, string P, Array frequence [len(∑)])
输入:目标字符串 T;模式字符串数组 P;字符使用频率数组 frequence。
功能:输出在 T 中出现的模式串。
Begin
  P_frequence="";//初始化模式字符串的频率字符串为空
  For i=1 to len(∑) do Shift[char(T[i])]=len(P);
  For i=0 to len(P)-1 do Shift[char(P[i])]=len(p)-1-i;
  For i=0 to len(P)-1 do
    For j=i to len(P) do
      If frequence[j]<frequence[i] then P_frequence[i]=P[j]
    End for
  End for;
  Pos_T=shift[P_frequence[0]];
  Do
    Pos_P=0;
    Shift=false;
    Do
      If T[Pos_T]<>P_frequence[Pos_P] then
        Pos_T=Pos_T + shift[char[P_frequence[Pos_P]]];
        Shift=truse;
      End if;
      Pos_P=Pos_P + 1;
    Until p=len(P)-1 or shift=true;
  Until len(T)-pos_T<len(P)
  Split(T);
End.
```

在网络的流量中，绝大部分的报文属于正常报文，所以包含入侵特征串的情况总体上在少数。基于这种事实，有没有可能在最优情况下将匹配的时间复杂度 $O(n/m)$ 进一步降低呢？

split_BM_in_frequence_match 算法的核心思想是：在引入字符使用频率启发跳跃的 BM_in_frenquence_match 算法基础上，进一步减少字符比较的次数，具体思想是利用目标字符串中不在模式字符串里出现而且相对居中的字符（简称切分字符）将目标字符串一分为二，对于被切分的字符串如果长度小于 m，则去掉（因为目标字符串的长度 n 是不小于 m）；对长度大于模式字符串的子串继续切分（如果可以继续切分的话），对长度等于或不能继续切分的长度大于模式串的切分子串调用基于使用频率启发的匹配算法进行匹配；对左右切分子串的继续重复以上操作，直到找到模式子串或者没有长度不小于模式字符串长度的切分子串。

具体算法描述如下：

```
函数 split_BM_in_frequence_match(string T, string P)
输入:目标字符串 T;Shift 数组;模式字符串数组 P。
功能:输出在 T 中出现的模式串。
Begin
  For i=1 to len(∑) do Shift[char(T[i])]=len(P);
  For i=0 to len(P)-1 do Shift[char(P[i])]=len(p)-1-i;
Split(T);
End.
函数 split(string Target)
输入:目标字符串 Target。
功能:输出在 Target 中出现的模式串。
Begin
  PosTarget = len(Target)+2 mod 2;
  If shift[char(t[PosT])]=len(P) then
    Leftstr = substring(Target,0,Pos-1);
    If len(Leftstr)<len(P) then
      return;
    Else
      CreateNewThread(split(Leftstr))//创建一个新线程并执行函数 split(Leftstr);
    end if;
    Rightstr = substring(Target,Pos+1,len(Target)-1);
    If len(Rightstr)<len(P) then
      return
    Else
      CreateNewThread(split(Rightstr))//创建一个新线程并执行函数 split(Rightstr);
    end if;
  else
    BM_in_frequence(Target,P);
  end if
end.
函数 BM_in_frequence(string Target, string Pattern)
输入:目标字符串 Target;模式字符串 Pattern;字符出现频率数组 frequence[len(∑)]。
功能:输出在 Target 中出现的模式串;
Begin
  PosTarget = (len(Target) mod 2)+1;
  i=1;
  Do while shift[char(Target[PosTarget])]<len(Pattern)/* 寻找相对位于字符串中央的切分字符 */
    PosTarget=PosTarget -i;
    If shift[char(Target[PosTarget])]<len(Pattern) then
      PosTarget=PosTarget + i
      If shift[char(Target[PosTarget])]<len(Pattern) then
        i=i+1;
        continue;
      else
        break;
      End if
    Else
      Break;
    End if
  End while
  If shift[char(Target[PosT])]=len(Pattern) then
    Leftstr = substring(Target,0,Pos-1);//取出左切分子串
    If len(Leftstr)<len(Pattern) then
      return
    Else
      If len(Pattern)<len(Leftstr)<len(Pattern) * 2 then
          Return BM_in_frequence(Leftstr,Pattern);
        else
          CreateNewThread(split(Leftstr))//创建一个新线程并对左切分子串执行函数 split(Leftstr);
      End if
    end if;
    Rightstr = substring(Target, PosTarget +1, len(Target)-1);//取出右切分子串
    If len(Rightstr)<len(Pattern) then
      return
    Else
      If len(Pattern)<len(Rightstr)<len(Pattern) * 2 then
          Return BM_in_frequence(Rightstr,Pattern);
        else
          CreateNewThread(split(Rightstr))//创建一个新线程并对右切分子串执行函数 split(Rightstr);
        End if
    end if;
  else
    BM_in_frequence(Target,Pattern);
  end if
end.
```

4　算法性能分析

由以上算法可知，改进算法与 BM 算法相比具有以下优点：

① 新方法在某些情况下比 BM 算法要优越得多。例如目标字符串 T="ebcdebcdebcdebcdebcdebcdebcd"，模式字符串 P="abcd"。首先根据模式串 P 构造针对每个字符向右跳跃距离 Shift 数组 Shift[hash(a)]= 3，Shift[hash(b)]=2，Shift[hash(c)]=1，Shift[hash(d)]=0，其他字符的跳跃距离均为 4，均不在模式字符串中出现，属于切分字符。在目标字

符串中 e 为切分字符，去掉 e 后的切分子串集合为{"bcd","bcd","bcd","bcd","bcd","bcd","bcd"}，这些切分子串的长度(等于 3)均小于模式字符串的长度(等于 4)，所以不用再进行字符串的比较。而如果按照 BM 算法进行匹配需要比较字符 4×7=28 次，移动比较位置 7 次。采用本文的新算法则只比较 7 次左右判断切分字符，没有字符移动操作。在 BM 算法最优情况执行时间近似相等，如 T="slkjdfsajdlsjhhkjhsaklhashedsekshfkskajdaj"，$n=\text{len}(T)=60$，p="mmmmnnnnn"，$m=\text{len}(p)=8$，BM 算法作字符比较 $n/m=7$ 次，移动比较位置 $n/m=7$ 次，采用新算法作切分 1+2+4=7 次，没有字符比较操作。

② 新算法便于并行编程，提高算法具体实现程序的执行速度。BM 算法中字符比较和移位操作只能串行操作，总的时间花销在最佳情况下为 $O([n/m])$，而新算法可以将切分操作并行进行，这样，实现总的时间开销最佳情况下为 $O(\log_2[n/m])$，在 $[n/m]$ 较大情况下，时间差别将很大，比如 $[n/m]=256$，则 $\log_2[n/m]=8$，BM 算法的执行时间是改进算法的 32 倍。

5 算法测试及结果

为了评测该算法的性能，随机地抽取一段文本和模式串，并在同一台计算机上用不同的算法进行匹配。测试文本 T="From automated teller machines and atomic clocks to mammograms and semiconductors, innumerable products and services rely in some way on technology, measurement and standards provided by the National Institute of Standards and Technology"，模式串 P="products and services"。分别用 BM 算法和 split_BM_in_frequence_match 算法在同一台计算机上进行匹配计算，并统计每种算法匹配时总的字符匹配次数。测试结果如表 4。

表 2 匹配算法与 BM 算法比较实验结果

	BM	split_BM_in_frequence_match
一次匹配的总的字符匹配次	129	109

实验结果表明了 split_BM_in_frequence_match 算法能取得比 BM 算法更快的速度。

结束语 本文研究了当前使用非常广泛且性能较好的 BM 算法，提出了基于字符使用频率和分治思想的字符串模式匹配 split_BM_in_frequence_match 算法，算法结合 BM 算法并利用字符统计使用频率和分治思想来提高坏字符的失配率，减少不必要的比较次数。同时，split_BM_in_frequence_match 算法更容易并行编程实现，可以进一步提高匹配的速度。比较试验进一步证实了 split_BM_in_frequence_match 算法的有效性。

参 考 文 献

[1] Knuth D, Morris J, Pratt P. Fast pattern matching in strings. SIAM Journal on Computing, 1997, 6(2): 323

[2] Boyer R S, Moore J S. A fast string searching algorithm. Communications of the ACM, 1997, 20(10): 762

[3] Nigel H R. Practical fast searching in strings. Software Practice and Experience, 1980, 10(6): 501-506

[4] Aho A, Corasick M. Efficient string matching: An aid to bibliographic search. Communications of the ACM, 1975, 18(6): 333-343

[5] http://crypt.math.fju.edu.tw/key/09/Freq.html

(上接第 157 页)

3.2 概念和物元的可满足性问题

定义 12 某个概念 C 或物元 W 在 MDDL 结构中是可满足的，当且仅当 C 或 W 不包含于 $\perp$。

上述定义的语义为，若在 MDDL 结构中存在一个解释 I 使得 $C^I\neq\emptyset$ 或 $W^I\neq\emptyset$，则称概念 C 或物元 W 是可满足的。

对于概念或物元的可满足性问题，关键就是要构造出关于概念或物元的一个解释，使得该解释是非空的，即不包含冲突。

以物元 W 为例，令 W 为一个要检测的物元，并假定某个体 a 为物元 W 的实例，于是得到断言公式 $W(a)$。现在令断言公式集 $A'=\{W(a)\}$，利用实例断言集的一致性检测算法来判断 A' 是否一致。如果 A' 中不包含冲突，则 A' 是一致的。此时，实际上就已经找到了物元 W 的一个解释 W^I，使得至少有 $a\in W^I$，即物元 W 不包含于 $\perp$，于是物元 W 是可满足的。否则如果 A' 是不一致的，则说明 W 包含于 $\perp$，此时 W 是不可满足的。

由上可知，概念或物元的可满足性问题可以直接转化为实例断言集的一致性问题，所以概念和物元的可满足性问题是可判定的。

结束语 本文针对动态描述逻辑在描述概念的属性及其变化时的不足，提出了一种带物元的动态描述逻辑 MDDL。该描述逻辑采用物元来统一描述概念和个体、属性、属性值，并按照传统描述逻辑的语义解释方法给出了物元的语义解释。这种描述事物的方法不仅充分表达了事物与属性之间的密切关系，而且可以体现执行动作时个体、属性和属性值的动态变化，使人们在处理问题的时候既考虑量的变化，又考虑质的变化。另外，描述逻辑 MDDL 中还利用物元"一物多征"的发散推理规则，扩充动态描述逻辑 DDL 的算法规则，提出一种新的 Tableau-M 算法，并根据该算法深入研究了 MDDL 的基本推理问题，即实例断言集的一致性检测和概念与物元的可满足性检测。由于物元的发散规则是为了模仿人类的发散性思维而建立的，因此利用它可以对现有知识库进行扩展，从而得到更完备的知识。

参 考 文 献

[1] Baader F, Nutt W. Basic description logic[G]// Baader F, et al. eds. The Description Logic Handbook: Theory, Implementation and Applications. Cambridge University Press, 2003: 47-100

[2] Baader F, Horrocks I, Sattler U. Description logics as ontology languages for the semantic Web// Dieter Hutter, Werner Stephan, eds. Festschrift in honor of Jörg Siekmann, Berlin: Springer, 2003

[3] 史忠植，董明楷，蒋运承，等. 语义 Web 的逻辑基础[J]. 中国科学，E 辑，2004，34(10)：1123-1138

[4] 蔡文，杨春燕，林伟初. 可拓工程方法[M]. 北京：科学出版社，2000

[5] 刘巍，张秀芳. 基于可拓信息的知识表示[J]. 系统工程理论与实践，1998，18(1)：104-107

[6] 蔡文，杨春燕，何斌. 可拓逻辑初步[M]. 北京：科学出版社，2003

[7] Lutz C, Sattler U. A Proposal for Describing Services with DLs // Horrocks T S, ed. Proceedings of the 2002 International Workshop on Description. Logics, Aachen: CEUR-WS, 2002: 129-140

[8] Baader F, Sattler U. An Overview of Tableau Algo-rithms for Description Logics. Studia Logica, 2001, 69: 5-40

[9] Schmidt-Schauβ M, Smolka G. Attributive concept description with complements. Artificial Intelligence, 1991, 48(1): 1-26