QuickSearch

Ein schneller Algorithmus zur Mustersuche in Zeichenfolgen

Bernhard Ziegler

Universität Stuttgart

Zusammenfassung. Quicksearch ist eine neue Variante des Algorithmus BoMo von Boyer und Moore zur Mustersuche in Texten. Sie ist bei Texten in natürlichen Sprachen, das heißt solchen mit relativ großen Alphabeten, und bei der Suche nach relativ kurzen Mustern den bisher bekannten BoMo-Varianten ebenbürtig, übertrifft sie aber bei langen Mustern und kleinen Alphabeten, wie sie bei der Suche nach Genetischem Code vorkommen, um einen Faktor von ca. 1.25 bei der Musterlänge 10 und bis zu einem Faktor > 2 bei Mustern ab der Größe 40.

Schlüsselwörter: Wortsuche, Mustererkennung, Boyer-Moore

Summary. QuickSearch is a new variant of BoMo, the Boyer-Moore algorithm for pattern matching in strings. For strings with relatively large alphabets such as natural-language texts and for short patterns, it matches with presently known BoMo variants; for long patterns and short alphabets as do occur in Genetic Code, it surpasses these variants by a factor 1.25 at pattern length 10, and up to a factor > 2 for patterns of length greater 40.

Key words: String Searching, Pattern matching, Boyer-Moore

Computing Review Classification: F.2.2, H.3.3

---

1 Einleitung

Mustersuche in Texten ist eine häufige Aufgabe in der Datenverarbeitung. Sie tritt auf in Editoren und bei Literaturrecherchen. In letzter Zeit ist noch die Suche in Genetischem Code dazugekommen. Texte in Genetischem Code sind meistens extrem lange Zeichenketten aus nur vier etwa gleich häufigen Zeichen. Seit 1977 ist mit BoMo, dem von Boyer und Moore [4] gefundenen Algorithmus, ein ausgezeichnetes Verfahren zur Lösung der genannten Aufgabe bekannt. Zahlreiche Autoren haben noch in den letzten Jahren neue Varianten vorgeschlagen, die unter bestimmten Bedingungen die Suchzeiten von BoMo verringern [1, 2, 8, 9]. Nachdem unsere Vergleichstests von und mit Quick Search fast beendet waren, erschien eine Arbeit von Hume und Sunday [6]. Die Autoren haben die einzelnen Algorithmen in voneinander unabhängige Bauteile zerlegt und deren Laufzeiten in einer umfangreichen Versuchsreihe mit unterschiedlichen Datensätzen auf Rechnern verschiedener Architektur miteinander verglichen, so daß sich jetzt jeder Anwender aus den Bausteinen den für seine Daten und seinen Rechner jeweils optimalen Algorithmus zusammensetzen kann. Auf diese Bausteine wird im nächsten Kapitel eingegangen.

Bisher wird in der die Effizienz von BoMo wesentlichen bestimmenden schnellen „Skip-Schleife“ nach einer Verschiebung des Musters über den Text die Vorgeschichte „vergessen“. Hier wird eine Methode vorgeschlagen, welche die Information des letzten Schrittes dazu verwendet, beim folgenden Schritt eine größere Verschiebung zu erhalten.

Bei unseren Tests hat sich gezeigt, daß die neue Methode bei Texten aus natürlichen Sprachen den schnellsten BoMo-Varianten schon bei kurzen Mustern ebenbürtig ist, sie aber bei Genetischem Code deutlich schlägt. Wir haben Schritte (Textzugriffe, Vergleiche, Verlassen der Skip-Schleife,...) gezählt, aber auf systematische Zeitmessungen mit optimierten Implementierungen, etwa in C, fast durchweg verzichtet. Wir haben angenommen, daß über die Häufigkeit der Zeichen in den Texten nichts bekannt ist. Deshalb haben wir nur solche BoMo-Varianten betrachtet, die solches Wissen nicht verwenden.

Im Abschnitt „BoMo-Bausteine und Nomenklatur“ wird die verwendete Notation erläutert und aufgezeigt, in welche Bausteine man BoMo zerlegen kann. Danach wird eine neue zweiparametrige Skip-Funktion eingeführt und erklärt, die größere Verschiebungen des Musters über den Text hinweg zuläßt. Schließlich werden neue Shift-Funktionen vorgestellt.

In den folgenden Abschnitten werden die Skip-Schleifen und Shift-Funktionen angegeben, mit denen die neuen Varianten sich messen müssen. Wir beschreiben die Daten und die Meßverfahren, berichten zusammenfassend über die Resultate und ziehen schließlich Folgerungen, wie die einzelnen Bausteine zu, dem Verwendungszweck angepaßten, BoMo-Varianten zusammengesetzt werden sollten. Im Anhang wird ein komprimierter Auszug unserer Meßresultate angegeben.

2 BoMo-Bausteine und Nomenklatur

Im Folgenden wird angenommen, daß der zu durchsuchende Text der Länge *Tl* in einem Feld *Text* [*] und das zu suchende Muster der Länge *Pl* in einem Feld *Pat* [*] geeigneter Größe gespeichert sind. Im Text sollen jeweils alle sich nicht überlappenden Exemplare des Musters gesucht werden. Wird ein Muster gefunden, so wird dies durch den Aufruf einer Prozedur *Locout* (*Index*) angezeigt. Ihr wird als Parameter der Index der Stelle mitgegeben, an der das Muster beginnt. Soll auf eine der angegebenen Bedingungen verzichtet werden, so muß die entsprechende BoMo-Variante nur geringfügig geändert werden.

Die an einigen Stellen verwendete Notation [*I*...*J*] bedeutet: alle Indizes *k* von *I* bis *J* ($I \leq k \leq J$). So steht z.B. *Pat* [*Pl*–1...*Pl*] = *Text* [*Ke*–1...*Ke*] statt *Pat* [*Pl*–1] = *Text* [*Ke*–1] & *Pat* [*Pl*] = *Text* [*Ke*].

Mit *K*0 und *Ke* werden die Stellen im Text indiziert, die *Pat* [0] und *Pat* [*Pl*] gegenüberliegen. In redundante runde Klammern gesetzte Ausdrücke könnten als Invariante einer eigenen Variablen zugewiesen werden, sodaß sie nur einmal zu berechnen sind, oder sie ließen sich in eine Funktionsdefinition mit aufnehmen. Diese Vereinfachungen haben wir zum Teil, des besseren Verständnisses wegen, weggelassen.

Wir verzichte hier darauf, auf die oben genannte Literatur verweisend, detailliert das Prinzip zu beschreiben, nach dem BoMo funktioniert. Die von Boyer und Moore *Delta*0, *Delta*1 und *Delta*2 genannten Funktionen werden hier *AL*, *A*0 und *D* genannt. Ist bei einer BoMo-Variante eine „Bremse" nötig, um das Textende zu bemerken, so wird hier angenommen, ein zusätzliches Exemplar des Musters sei an das Ende des Textes angehängt worden. Dadurch wird die Zahl der Tests „Textende überschritten" stark reduziert. Dafür muß das Muster einmal zusätzlich erkannt, aber nicht gemeldet werden. Soweit uns bekannt, wurde diese Art Bremse bisher noch nicht bei BoMo-Varianten verwendet. Unter diesen Voraussetzungen läßt sich die Originalversion von BoMo so formulieren:

```
BoMoOriginal;
K := Pl;
loop
   (*** skip loop ***)
   while K <= Tl do K :=  K + AL[Text[K]] od;
                     (* Skip-Funktion: AL  *)
 if K <= Large       (* text exhausted,    *)
  then exit fi;      (* quit loop!         *)
   K := K - (Large + 1);  J := (Pl - 1);
   (*** test equal part ***)
   while (Text[K] = Pat[J]) & (J > 0)
     do K := K - 1;  J := J - 1 od;
   if J = 0
     then            (* pattern matched    *)
       Locout(K + 1);  K := K + (2 * Pl);
     else            (* no match           *)
   (*** shift part ***)
       K := K + max(A0[Text[K]], D[J]);
    fi;              (* Shift: max(A0, D)  *)
 end loop
end (* BoMoOriginal *);
```

Die angegebene Einteilung von BoMo in drei im wesentlichen voneinander unabhängige Bausteine, aus denen sich ein passender Algorithmus zusammenbauen läßt, wurde von Hume und Sunday in [6] erstmals herausgestellt. Die drei Teile lassen sich getrennt optimieren. Wenn ein Zeichen bekannt ist, das nicht im Text vorkommt, läßt sich nach [7] der Vergleich $J > 0$ eliminieren. Damit sind die Optmierungsmöglichkeiten von „test equal" unter den oben angegebenen Voraussetzungen (Zeichenstatistik unbekannt) erschöpft.

3 Eine neue Variante der Skip-Funktion A

Die Skip-Schleife wird in BoMo stets dann verlassen, wenn gilt: *Text* [*Ke*] = *Pat* [*Pl*]. Danach wird in natürlichen Sprachen der „test equal"- und der „Shift-Teil" in den meisten Fällen wegen *Text* [*Ke*–1] $\neq$ *Pat* [*Pl*–1] nur einmal durchlaufen. Gelingt es, die Skip-Schleife seltener zu verlassen, so wird die Effizienz steigen.

Ein erster Ansatz, dieses Ziel zu erreichen, war der, statt des letzten ein anderes Zeichen in der Skip-Schleife zu verwenden, das im Text seltener vorkommt [5]. Das setzt voraus, daß etwas über die Häufigkeit der Zeichen des Textes bekannt ist. Außerdem ist zu prüfen, ob der Verlust an Verschiebung in der Skip-Schleife tatsächlich durch den Gewinn aufgewogen wird, der entsteht, wenn der „test equal"-Teil seltener durchlaufen wird.

Bisher wurde noch nicht untersucht, ob es sich lohnt, einen Teil der Informationen, die die Vorgeschichte liefert, zu speichern und in die Berechnung der nächsten Verschiebung mit einzubeziehen. Dabei bietet es sich an, die Skip-Schleife erst dann zu verlassen, wenn die beiden letzten Zeichen des Musters zu denen des Textes passen (*Pat* [*Pl*–1...*Pl*] = *Text* [*Ke*–1...*Ke*]). Beides ist hier erstmals implementiert und getestet worden.

Boyer und Moore haben die Skip-Funktion *A* definiert als:

$$A(B) = \min(S : B = Pat\,[Pl\text{–}S\,]), \text{ falls } B \neq Pat\,[Pl\,],$$
$$= Large, \text{ falls } B = Pat\,[Pl\,] \text{ mit } Large \geq Pl\text{+}Tl.$$

Anmerkung: Hier wird angenommen, daß jedes virtuelle Zeichen *Pat* [*J*] mit $J < 1$ jede geforderte Eigenschaft besitzt.

Bezieht man die Größe *l* der letzten Verschiebung in die Berechnung der folgenden Verschiebung mit ein, so ergibt sich die von zwei Parametern abhängende Skip-Funktion *AA*(*B, l*) als:

$$AA(B,l) = \min(S : B{=}Pat\,[Pl\text{–}S],\ Pat\,[(Pl\text{–}l)\text{–}S]{=}Pat\,[Pl\text{–}l]):$$
$$1 \leq l \leq Pl,\ B \neq Pat\,[Pl\,].$$

Hier bedeutet *S* nicht die Verschiebung des Musters, sondern die Änderung des Zeigers auf den Text. *AA* soll auch bewirken, daß die Skipschleife erst dann verlassen wird, wenn gilt:

Text [*Ke*–1 ... *Ke*] = *Pat* [*Pl*–1 ... *Pl*].

Um das zu erreichen, muß eine negative Verschiebung für den Fall *Text* [*Ke*] = *Pat*[*Pl*] vorgesehen werden. Da diese Verschiebung wiederum ein Parameter von *AA* ist, muß auch für diesen Parameter die Verschiebung berechnet werden. Daher ist die Definition von *AA* zu ergänzen um:

Zeichen	Verschiebung -1	0	1	2	3	4	5	6	7	8	9	10	11
c	11	0	*Large* -3, 0	-1	-1		-1			-1	-1	-1	-1
a	*Large*–1 -2, 1		9	3	3		3			3	1	1	1
b	5		2	8	8		5			2	2	2	2
#	11		10	10	10		10			10	10	10	10

Abb. 1. Funktionswerte von *AAL*, *AA* und *AA*0 am Beispiel *abcabcabac*

$AA(Pat\,[Pl\,], l) = -1 : l \neq -1,$

$AA(B, -1) =$

min (*S* : *Pat* [*Pl*–1–*S*...*Pl*–*S*] = *Pat* [*Pl*–1...*Pl*]) + 1:
$B \neq Pat\,[Pl - 1],$
(* Ausgleich der negativen Verschiebung *)
Deshalb gibt es hier auch eine Verschiebung $S = Pl + 1$.

$AA(B, Pl + 1) = AA(B, Pl\,)$

$AA(Pat\,[Pl - 1], -1) = Large - 1$

(* Abbruchbedingung der Skip-Schleife *)
Hier wurde zunächst *Pat* [*Pl*] = *Text* [*Ke*] erkannt und erst danach die Bedingung: *Pat* [*Pl*–1] = *Text* [*Ke*–1].

$AA(Pat\,[Pl\,], 1) = Large$

(* Abbruchbedingung der Skip-Schleife *)
Hier wurde zunächst *Pat* [*Pl*–1] =*Text* [*Ke*–1] erkannt und erst danach *Pat* [*Pl*] = *Text* [*Ke*]. Die beiden Abbruchbedingungen müssen sich unterscheiden, damit nach dem Verlassen der Skip-Schleife im Text richtig aufgesetzt wird.

Je nachdem, wie die Skip-Schleife implementiert wird, ist in *AA* zu ersetzen: (*Large*,*Large*–1) durch (-3, -2) oder durch $(0, 1)$. Die zugehörenden Skip-Funktionen heißen *AAL*, *AA*, *AA*0, und es gilt *AA*0[*Pat*[*Pl*], 0] $= 0$. Bei der Verwendung von *AA*0 ist die Skip-Schleife ausrollbar. Bei der angegebenen Version von *AA*0 endet die Indexveränderung, wenn zunächst *Text* [*Ke*–1] = *Pat* [*Pl*–1] und danach erst *Text* [*Ke*] = *Pat* [*Pl*] gegolten hatte, d. h. der Index bleibt auf die Stelle *Ke* positioniert, sodaß in einem Teil der Fälle daher zweimal auf *Text* [*Ke*] zugegriffen wird.
Wie man leicht sieht, gilt stets:

$$AA\;(B, S) \geq A(B)$$

für jedes *S* und jedes *B*, $B \neq Pat\,[Pl\,]$. Das heißt, *AA* verschiebt nie weniger als *A*. Außerdem wird die Skip-Schleife seltener verlassen. Es muß allerdings auf ein zweidimensionales Feld zugegriffen werden, dessen Ansteuerung länger dauern kann als die des eindimensionalen. Die Unterschiede in der Verschiebung sollen am Beispiel *abracadabra* dargestellt werden:

```
Text      ..........b.r..........
                    #
Muster  abracadabra
                    = #
Muster    abracadabra              A = 2, AA = 2
                    # =
Muster     abracadabra             A = 1
                    # =
Muster          abracadabra
                    = =
Muster          ...abracadabra        AA = 11
```

Hier sollen die Funktionswerte von *AAL*, *AA* und *AA*0 in einer Tabelle (*Abb.* 1) am von Knuth vorgeschlagenen Beispiel *abcabcabac* dargestellt werden. Dabei steht # als Symbol für jedes im Muster nicht vorkommende Zeichen. Die Verschiebungen und damit die Spalten (0), 4, 6 und 7 werden nie erreicht.
Bei der folgenden BoMo-Variante wird neben der Skip-Funktion *AA* auch eine nur vom Muster abhängige invariante Shift-Funktion verwendet, die im Abschnitt „neue Varianten der Shift-Funktion“ beschrieben wird.

```
BoMoAA         (* BoMo mit Shiftfunktion AA, *)
               (* eine QuickSearch-Variante  *)
K := Pl;
loop
   S := Pl;
   while Ke <= Tl
    do S := AA[Text[Ke], S];
       Ke := Ke + S
     od;
 if K < Large (* text exhausted, quit loop! *)
  then exit fi;
   Ke := Ke - Large; K := Ke - 2;
   J := (Pl - 2);
   while Text[K] = Pat[J] & J > 0
    do K := K - 1;   J := J - 1 od;
   if J = 0
     then      (* pattern matched            *)
       Locout(K + 1);  Ke := Ke + Pl
     else      (* no match                   *)
       Ke := Ke + CCShift
    fi;
 end loop;
end (* BoMoAA *);
```

Hume und Sunday haben bei ihren Untersuchungen über die Laufzeiteffizienz [6] BoMo-Varianten verwendet, bei denen die Skip-Schleife ausgerollt (unrolled) wurde, d. h. mehrere Schleifendurchläufe wurden durch ein gleichwertiges gestrecktes Programm ersetzt, um die Anzahl der Tests auf die Schleifenabbruchbedingung zu verringern, etwa so:

```
while K <= Tl
do
  {1} K:= K + A0[Text[K]];    (* AO=AL, aber  *)
  {2} K:= K + A0[Text[K]];    (* A0[Pat[Pl]]  *)
  {3} K:= K + A0[Text[K]];    (* = 0, statt   *)
  {4} K:= K + AL[Text[K]];    (* Large        *)
 od;
```

Die Berechnung der Funktion *AA* ist so aufwendig, daß man auf die Berechnung zweier Felder verzichtet und nur *AA*0 (mit 0 an der Stelle von *Large*) berechnet. Der Abbruch der Skip-Schleife ist nicht mehr mit *Large* erzwingbar. Deshalb wird an den Text ein Muster als Bremse angehängt. Damit ergibt sich folgende ausrollbare QuickSearch-Variante:

```
BoMoAA0         (* ausrollbar              *);
K := 0;
loop S := Pl;
   while S # 0
    do K := K + S;
       S := AA0[Text[K], S]
     od;
   J := (Pl - 2); Ke := K; K := K - 2;
   while Text[K] = Pat[J] & J > 0
    do K := K - 1; J := J - 1 od;
   if J = 0
    then                   (* pattern matched *)
 if K >= (Tl - Pl)         (* text exhausted, *)
  then exit fi;            (* quit loop!      *)
      Locout(K + 1); K := Ke
    else                   (* no match        *)
      K := Ke + (CCShift - Pl)
    fi;
 end loop;
end (* BoMoAA0 *);
```

4 Neue Shift-Funktionen

Scheitert der Vergleich von Text und Muster, so kann auf jeden Fall das Muster wenigstens um eine von ihm abhängige Invariante *CShift* verschoben werden. *CShift* gibt an, wie weit vom rechten Rand des Musters das Zeichen *Pat*[*Pl*] zum zweiten Mal auftaucht. *CShift* ist definiert durch:

$$CShift = \min(S : Pat\ [Pl–S\ ]=Pat\ [Pl\ ],\ S > 0).$$

Wenn $CShift \geq Pl - 1$ ist, so gibt es keine Funktion mit einer größeren Verschiebung. Im anderen Fall lassen sich *CShift* und die Funktion *A*0 zu einer Funktion *ACS*0(*B*) kombinieren, die so definiert ist:

$$ACS0(B) =$$
$$\min(S : B{=}Pat[Pl–S\ ],\ Pat[(Pl–CShift)–S\ ]{=}Pat[Pl])$$
$$B = Text\ [\mathrm{Ke} + CShift\ ].$$

Analog dazu läßt sich verfahren, wenn beim Verlassen der Skip-Schleife die zwei letzten Zeichen gepaßt haben. Es gilt:

$$CCShift =$$
$$\min(S : Pat[(Pl–1)–S...Pl–S]{=}Pat[(Pl–1)...Pl])$$

und

$$ACCS0(B) =$$
$$\min(S : Pat[Pl–S]{=}B,\ Pat[(Pl–1–CShift)–S]{=}Pat[Pl–1],$$
$$Pat[(Pl–CShift)–S]{=}Pat[Pl]);\ B = Text[Ke{+}CCShift].$$

Damit ergibt sich die neue QuickSearch-Variante:

```
BoMoAA0ACCS0               (* ausrollbar     *);
K := 0; S := Pl;
loop
   while S # 0
    do K := K + S; S := AA0[Text[K],S] od;
   J := (Pl - 2);   Ke := K;   K := K - 2;
   while Text[K] = Pat[J] & J > 0
    do K := K - 1;   J := J - 1 od;
   if J = 0
    then                   (* pattern matched *)
 if K >= (Tl - Pl)         (* text exhausted, *)
  then exit fi;            (* quit loop!      *)
      Locout(K + 1);   K := Ke;   S := Pl;
    else                   (* no match        *)
      K := Ke + CCShift;
      S := ACCS0[Text[K]];
    fi;
 end loop;
end (* BoMoAA0ACCS0 *);
```

5 Varianten der Skip-Schleife

Wir haben beim Test folgende von uns jeweils mit gleicher Sorgfalt optimierte Skipschleifen verwendet, die möglichst wenige zu zählende Operationen enthalten:

```
while (Pat[Pl]) = Text[Ke]
 do Ke := Ke + AS[Text[Ke+1]] od;
```

Diese Variante mit $AS\ [B\ ] = AL\ [B] + 1$ ist [9] entnommen.

```
while B # (Pat[Pl])
 do Ke := Ke
    + max(A0[Text[Ke], A0[Text[Ke+1]]+1);
    B := Text[Ke]
  od;
```

Diese Variante stammt von Smith [8]. Wir haben, um die Berechnung der Maxima zu vermeiden, statt dessen deren Werte a priori berechnet und in einem zweidimensionalen Feld abgespeichert.

```
while Ke <= Tl
 do Ke := Ke + AL[Text[Ke]] od;

while S # 0
 do Ke := Ke + S; S := A0[Text[Ke]] od;
```

Diese beiden Schleifen entsprechen dem originalen BoMo. Dabei ist die zweite Version ausrollbar.

```
while K <= Tl
 do S := AAL[Text[K],S]; K := K + S od;

while S > -2
 do K := K + S;  S := AA[Text[K],S] od;

while S # 0
 do K := K + S; S := AA0[Text[K],S] od;
```

Diese drei Schleifen sind Versionen der Skipschleife von Quick-Search. Die dritte davon ist ausrollbar.
Nach dem Verlassen der Skip-Schleife haben wir in allen Versionen an der Stelle *Ke* + 1 wieder aufgesetzt, wenn kein Muster gefunden wurde.

6 Die Shift-Funktionen

Wir haben folgende Shift-Funktionen nach dem „test equal"-Teil verwendet:

- *CShift*, *CCShift*: Invarianten, die sich indirekt aus Horspools Arbeit [5] erschließen lassen.
- max(*A*0[*Text* $[K]], D[J]$): Das ist die Verschiebung, die Boyer und Moore in ihrer Arbeit verwendet haben.
- $D[J]$: Version ohne Berechnung des Maximums und ohne Zugriff auf den Text.
- *DD* [*Text* $[K], J]$: Diese Funktion kombiniert *A* und *D* zu einer Funktion mit zwei Parametern. Diese Funktion wurde ebenfalls schon 1977 von Boyer und Moore vorgeschlagen (und nicht von Giancarlo, wie Hume und Sunday fälschlich behaupten). Wir habe sie etwa 1980 von Studenten implementieren lassen. Sie reichte bei den damaligen Rechnern aufgrund der langsamen und kleinen Arbeitsspeicher nicht an die Effizienz der anderen Versionen heran.
- *ACS*0[*Text* [*Ke*]], *ACCS*0[*Text* [*Ke*]]: Die erste der beiden Funktionen wurde von uns in [3] vorgestellt, die zweite hier erstmals vorgeschlagen. Hierbei muß vor der Auswertung von *ACS*0 *Ke* := *Ke* + *CShift* und vor der von *ACCS*0 *Ke* := *Ke* + *CCShift* ausgeführt werden.

Alle Funktionen außer *CCShift* und *ACCS*0 wurden mit der gleichen Skip-Schleife getestet. Die beiden genannten Funktionen setzen voraus, daß *Pat* [*Pl*–1...*Pl*] = *Text* [*Ke*–1...*Ke*] gilt. Deshalb sind sie nicht mit den einparametrigen Skip-Funktionen kombinierbar, die nur *Pat* [*Pl*] = *Text* [*Ke*] garantieren.

7 Daten und Meßverfahren

Wir haben in einem ersten Test jedes Wort der Länge $Pl \geq 2$ aus der UNIX- Datei *usr/dict/words* als Muster verwendet und es in dieser Datei wieder gesucht. Für die Musterlänge 1 verwendet man besser einen eigenen Algorithmus, besonders weil Quick-Search diesen Sonderfall nicht einschließt. Die Wörter sind so verteilt, daß am häufigsten Wörter der Länge sieben vorkommen. Ab $Pl = 16$ wird die Zahl der Wörter so klein, daß die Statistik keine zuverlässigen Aussagen zuläßt.
Wenn die Verschiebung $CShift \geq Pl - 1$ ist, dann gibt es keine Funktion, die nach dem „test equal"-Teil eine größere Verschiebung als *CShift* bewirken kann. Bei den 25075 Wörtern war 19525 mal $CShift \geq Pl - 1$, d.h. in rund 78% aller Fälle. Dieser Mittelwert für die Wörter aller Längen dürfte bei kurzen Wörtern, bei denen Buchstabenwiederholungen selten sind, noch wesentlich größer sein. Damit wird klar, warum schon Horspool [5] beobachtet hat, daß die Shiftfunktionen nach dem „test equal"-Teil nur sehr wenig zur Gesamtverschiebung beitragen, sodaß sich ihre Berechnung nicht lohnt und die manchmal kleinere Verschiebung mit *CShift* durch den geringeren Aufwand beim Zugriff mehr als ausgeglichen wird. Mit wachsender Wortlänge und fallender Alphabetgröße wächst auch der Einfluß der Shiftfunktionen wieder.

Schon Boyer und Moore haben vorgeschlagen, als Maß für die Effizienz der Suchalgorithmen die Zahl der Zugriffe auf den Text zu verwenden. Andere Autoren haben lieber die Zahl der Vergleiche von Muster- mit Textzeichen verwendet, besonders dann, wenn dieses Maß ihnen einen Vorteil verschaffte. Es ist aber den BoMo-Varianten nicht angemessen. Kommt *Pat* [*Pl*] im Text nicht vor, so wird in BoMo niemals ein Text- mit einem Musterzeichen verglichen.
Um eine möglichst genaue Bewertungsgrundlage zu erhalten, haben wir folgende Operationen gezählt und ausgewertet:

- Textzugriffe
- einfach indizierte Feldzugriffe
- doppelt indizierte Feldzugriffe
- Feldzugriffe insgesamt
- Schlüsselvergleiche (= Vergleiche von Text- mit Musterzeichen)
- Indexvergleiche
- Indexoperationen
- logische Operationen
- Wechsel von Skip-Schleife zum „test equal"-Teil.

Anmerkung: Unter Index wird hier eine ganzzahlige Größe verstanden, mit der die Komponente eines Feldes adressiert wird. Auch eine reine Wertzuweisung gilt als Indexoperation.
Die Zahl der logischen Operationen blieb vernachlässigbar klein. Die Effizienz der Algorithmen haben wir außer durch die Zähler in Einheiten der Art (überstrichener Text)/(Zahl der entsprechenden Operationen) gemessen. Dabei haben wir auch alle Feldzugriffe und alle Vergleiche jeweils zusammen bewertet.
Während der Testphase unserer BoMo-Varianten erschien die Arbeit von Hume und Sunday [6]. Wir haben unsere Tests danach erweitert, unterlegene Varianten aber nicht mehr auf die neuen, in [6] angebotenen Daten angesetzt. Von diesen Daten haben wir nur die Dateien mit Genetischem Code verwendet. Dabei wurden je 200 vorgeschlagene Muster der Längen 10, 20, 30, 40, 50, 100, 150, 200 und 250 in der Code-Datei gesucht.

8 Resultate

8.1 *Die Skip-Funktionen*

Keine der Skip-Funktionen erweist sich bei allen gezählten Größen als überlegen. Die Smithsche Variante,

$$(\max(\ldots[Ke], \ldots[Ke+1]),$$

ist wegen der großen Zahl von Textzugriffen nicht konkurrenzfähig. Die Variante von Sunday kompensiert ihren Nachteil bei den Textzugriffen durch Vorteile bei den Indexoperationen. Zeitmessungen haben ergeben, daß die dreifach ausgerollte Version vor allem bei kurzen Mustern bis etwa zur Länge 9 eminent schnell ist. Die hier erstmals vorgestellten Skip-Funktionen sind bei kleinen Mustern ($Pl \leq 9$) den besten in den BoMo-Varianten ebenbürtig, übertreffen sie aber bei größeren Mustern. Sie sorgen dafür, daß die Skip-Schleife sehr viel seltener verlassen wird. Dadurch reduziert sich bei ihnen der Aufwand für die nach dem Verlassen der Schleife anfallenden Reorganisationsoperationen auf 1/6.
Bei Genetischem Code dagegen erweisen sie sich als haushoch überlegen. Bei allen Messung sind sie um einen Faktor von ca. 2 bei Mustern mit $Pl = 10$, bis zu einem Faktor > 5 bei Mustern mit $Pl \geq 100$ schneller als jeder Konkurrent.

8.2 *Die Shift-Funktionen*

Bei den Shift-Funktionen erweisen sich *CShift* und *CCShift* bei Texten in natürlichen Sprachen und Wörtern als Muster nicht zuletzt deshalb als erste Wahl, weil sie in ca. 80% der Fälle die maximal mögliche Verschiebung liefern und keinen zusätzlichen Aufwand für ihre Berechnung verlangen, da sie bei der ohnehin nötigen Erzeugung der Skip-Funktionen anfallen. Bei längeren Mustern, etwa ab $Pl = 10$, spielen die Funktionen *ACS*0 und *ACCS*0 ihre Vorteile aus. Sie sind beide Varianten der Skip-Funktion *A*, bei deren Berechnung ein Ausschnitt der Vorgeschichte die Größe der Verschiebung mitbestimmt. Einen noch größeren Teil der Vorgeschichte berücksichtigt die Funktion *DD* [Zeichen, Position].

Für die Suche in Genetischem Code sind weder die Invarianten *CShift* und *CCShift* noch die Funktionen *ACS*0 oder *ACCS*0 geeignet. Sie fallen gegen die Shift-Funktion von Boyer und Moore max(*A*0, *D*) stark ab. Die Funktion *DD* übertrifft vor allem bei längeren Mustern alle anderen bis zu einem Faktor 2.

9 Schlußfolgerungen

Wie aus den Meßresultaten des Anhangs hervorgeht, schneiden die QuickSearch-Varianten bei fast allen Zählungen am besten ab. Nimmt man das schon von Boyer und Moore vorgeschlagene Maß, die mittlere Verschiebung pro Textzugriff, als ausschlaggebend an, dann erweisen sich die hier vorgeschlagenen neuen Algorithmen mit den Skip-Funktionen *AA* bei Texten in natürlichen Sprachen und Mustern bis zur Länge 15 den bisher bekannten Varianten als ebenbürtig oder leicht überlegen. Außerdem wird bei ihnen die Skip-Schleife nur weniger als 1/6 so oft verlassen wie bei den anderen. Damit reduziert sich der Aufwand für den „test equal“- und den Shift-Teil um den gleichen Betrag. Ob das ausreicht, um den Mehraufwand für den Zugriff auf zweidimensionale Felder zu kompensieren, mag von Installation zu Installation verschieden sein.
Bei der Suche in Genetischem Code aber übertreffen die Quick Search-Varianten die BoMo-Varianten um Faktoren von 1.25, bei einer Musterlänge von 10, bis zu einem Faktor von etwas über 2, ab Mustern der Länge 40. Daraus ist zu folgern, daß die neuen BoMo-Varianten den alten Versionen stets dann überlegen sind, wenn entweder die Muster sehr lang oder die Alphabete klein sind oder beides zutrifft.
Die Shiftfunktionen tragen in natürlichen Sprachen meist nur wenig zur Gesamtverschiebung bei. Im Allgemeinen bietet es sich an, die bei der Berechnung der Skip-Funktion *A* ohnehin anfallende Invariante *CShift* zu verwenden. In natürlichen Sprachen gilt für die Invariante *CShift* bei den meisten Wörtern: $CShift \geq Pl - 1$. Es könnte sich lohnen, nach ihrer Berechnung fest auf maßgeschneiderte BoMo-Varianten zu verzweigen, etwa solche mit *ACS*0. Bei der Suche in Genetischem Code aber hat sich *DD* als allen anderen Shift-Funktionen überlegen gezeigt, da *DD* das Meiste an Vorgeschichte in die Berechnung der Verschiebung mit einbezieht. Deshalb wird sie in diesem Fall am besten mit einer der Skip-Funktionen von QuickSearch zu QuickerSearch zusammengesetzt. Diese Kombination dürfte immer dann optimal sein, wenn entweder, wie oben, die Muster sehr lang oder die Alphabete klein sind oder beides zutrifft.

10 Schlußbemerkungen

Ich bedanke mich bei Klaus Lagally für seine Hinweise beim Abfassen dieses Berichts. Udo Merkel verdanke ich wesentliche Unterstützung bei den Messungen. Er hat den Testrahmen geschrieben, die Meßdaten aufbereitet und mir beim Austesten der Algorithmen geholfen.

Literatur

1. Baeza-Yates, Ricardo A.: Improved string searching. Software - Practice and Experience 19.3, 257 - 271 (1989).
2. Baeza-Yates, Ricardo A.: String searching algorithms revisited. Lecture Notes in Comp. Sci. 382, Springer-Verlag (1989).
3. Baeza-Yates, Ricardo A., Krogh, Fred T., Ziegler, Bernhard, Sibbald, Peter R., Sunday, Daniel M.: Notes on a very fast substring search algorithm. In Technical Correspondence Comm. ACM 35.4, 132 - 137 (1992).
4. Boyer, Robert S., Moore, J. Strother: A fast string search algorithm. Comm. ACM 20.10, 762 - 772 (1977).
5. Horspool, R. Nigel: Practical fast fast searching in strings. Software - Practice and Experience 10.8, 501 - 506 (1980).
6. Hume Andrew., Sunday, Daniel.: Fast string searching. Software - Practice and Experience 21.11, 1221 - 1248 (1991).
7. Knuth, Donald E., Morris, James H., Pratt, Vaughan R.: Fast pattern matching in strings. SIAM J. Comput. 6.2, 323 - 350 (1977).
8. Smith, P. D.: Experiments with a very fast substring search algorithm. Software - Practice and Experience 21.10, 1065 - 1074 (1991).
9. Sunday, Daniel M.: A very fast substring search algorithm. Comm. ACM 33.8, 132 - 142 (1990).

Bernhard Ziegler
Universität Stuttgart
Institut für Informatik
Breitwiesenstr. 20-22
D-70565 Stuttgart

11 Anhang: Meßdaten

Wir haben zu unseren Messungen zwei verschiedene Datensätze verwendet, und zwar für die Suche in Texten

- natürlicher Sprachen Muster aus dem UNIX-Wörterbuch `/usr/dict/words`. Dabei haben wir einmal alle vorhandenen Wörter ab Länge 2, ein zweites Mal nur noch (falls vorhanden) jeweils 200 zufällig ausgewählte Wörter der Länge 2 bis 15 in einem String der Gesamtlänge 2.2 MB gesucht, den wir aus den UNIX-Dateien `/usr/man/man`*x*`/*` ($x = 1,2$) konkatenierten;

- Genetischen Codes die von Hume und Sunday in [6] zur Verfügung gestellten Daten, d.h. den String `dna.test` der Länge $\sim$ 1 MB sowie die vorgeschlagenen je 200 Wörter der Länge x, die wir den Dateien `dna.`*x* ($x = 10$, 20, 30, 40, 50, 100, 150, 200, 250) entnahmen.

Getestet wurden

- sieben zum Teil nur geringfügig voneinander abweichende Skip-Funktionen und Skip-Schleifen jeweils mit der selben Shift-Funktion $shift = 1$. Die bei den angegebenen Skip-Schleifen ermittelten Resultate stehen in der Tabelle unter folgenden Überschriften:

  BMOrig:
  ```
  while S > 0
    do Ke := Ke + S; S := A0[Text[Ke]] od;
  ```
  ausrollbare Form der Originalversion von Boyer und Moore [4];

  BMLarge:
  ```
  while Ke <= Tl
    do Ke := Ke + AL[Text[Ke]] od;
  ```
  Originalversion von Boyer und Moore [4].

  Sunday:
  ```
  while Text[Ke] # LPat
    do Ke := Ke + AS[Text[Ke+1]] od;
  ```
  Schleife nach Sunday [9], aber die Skip-Schleife wird aus Gründen der Effizienz getrennt behandelt. Außerdem gilt: `LPat == Pat[Pl]`.

  SmithO:
  ```
  while Text[Ke] # LPat
  do Ke:=Ke+max(A0[Text[Ke]],AS[Text[Ke+1]])
   od;
  ```
  Die Schleife wurde nicht in der angegebenen Form implementiert. Statt die Maxima jedesmal neu zu berechnen, wurden sie einmal in einer Tabelle gespeichert und jeweils von dort abgerufen, um an Effizienz zu gewinnen.

  BMZm:
  ```
  while S > -2
    do Ke := Ke + S; S := AA[Text[Ke],S] od;
  ```

  BMZUr1:
  ```
  while S # 0
    do Ke := Ke + S; S:= AA0[Text[Ke],S] od;
  ```

  BMZLarge:
  ```
  while Ke <= Tl
    do S := AA[Text[Ke],S]; Ke := Ke + S od;
  ```

  Die letzteren drei sind Varianten der hier eingeführten Skip-Schleife;

- sieben verschiedene Shift-Funktionen wie folgt

  BMH:
  ```
  shift = CShift
  ```
  Ergibt sich aus der Arbeit von Horspool [5], wenn die Skip-Schleife als separates Programmstück implementiert wird.

  BMO:
  ```
  shift = max(A0[Text[K]], D[J])
  ```
  Originalverschiebung von Boyer und Moore [4].

Pl	occurs	Skiptest: Zahl der Textzugriffe													
		BMOrig	%	*BMLarge*	%	*Sunday*	%	*SmithO*	%	*BMZm*	%	*BMZUr1*	%	*BMZLarge*	%
2	175	1189559	100	1189556	100	1597530	134	1554142	131	1170615	98	1172819	99	1170612	98
3	763	806668	100	806664	100	1217040	151	1168371	145	791838	98	793159	98	791834	98
4	2188	623687	100	623681	100	1003651	161	945400	152	606609	97	607817	97	606603	97
5	3169	509117	100	509110	100	856067	168	793685	156	493198	97	494360	97	493191	97
6	3864	436835	100	436827	100	757127	173	687964	157	419716	96	420877	96	419708	96
7	4060	381773	100	381764	100	675467	177	605328	159	364856	96	365790	96	364847	96
8	3614	340910	100	340899	100	611383	179	541428	159	323663	95	324533	95	323653	95
9	3088	309468	100	309457	100	563281	182	491414	159	291847	94	292667	95	291836	94
10	1971	283440	100	283428	100	522517	184	450730	159	265890	94	266660	94	265878	94
11	1119	263442	100	263429	100	488852	186	416648	158	245076	93	245773	93	245063	93
12	592	243104	100	243089	100	453324	186	385586	159	225804	93	226425	93	225790	93
13	280	229673	100	229658	100	430935	188	361916	158	211387	92	211982	92	211371	92
14	116	215838	100	215822	100	407473	189	339468	157	197455	91	197987	92	197439	91
15	44	202852	100	202835	100	384522	190	319348	157	185394	91	185838	92	185377	91
16	17	191738	100	191719	100	364302	190	301016	157	174231	91	174634	91	174213	91
17	7	184849	100	182279	99	351287	190	290787	157	168259	91	168666	91	165702	90
18	4	173406	100	170900	99	330545	191	272751	157	157885	91	158132	91	155394	90
20	1	156953	100	154735	99	293757	187	247667	158	142722	91	142888	91	140525	90
21	2	154496	100	152425	99	289123	187	241926	157	137137	89	137397	89	135074	87
22	1	135439	100	133421	99	256028	189	221171	163	126270	93	126270	93	124258	92

Abb. 2. Zugriffe auf Text natürlicher Sprache; $Tl = 2227990$; 25075 Muster

BM1D:
`shift = D[J]`
Auf den Vergleich mit $A0$ wird verzichtet → kein Textzugriff, kein Vergleich.

BM2D:
`shift = DD[Text[K], J]`
Zusammenfassung von A und D zu einer einzigen Funktion mit zwei Parametern.

BMSZ:
`shift = ACS[Text[Ke+CShift]]`
ACS wurde oben im Text beschrieben.

Beim Test der angegebenen Shift-Funktionen wurde stets die selbe Skip-Schleife (*BMOrig*) verwendet. Die beiden folgenden Shift-Funktionen sind nicht mehr direkt mit den übrigen vergleichhbar, da bei ihnen anstelle von *BMOrig* die Skip-Schleife *BMZm* verwendet wurde:

BMHZ:
`shift = CCShift`

BMZ:
`shift = ACCS[Text[Ke+CCShift]]`

Beide Shift-Funktionen wurden hier neu eingeführt.

Die angegebenen Tabellenwerte sind Mittelwerte für jeweils eine Musterlänge. Der Text wird dabei bis zum Ende durchsucht. Der Aufwand zur Berechnung der Tabellen blieb unberücksichtigt.

Pl	occurs	Skiptest: Zahl der Textzugriffe													
		BMOrig	%	*BMLarge*	%	*Sunday*	%	*SmithO*	%	*BMZm*	%	*BMZUr1*	%	*BMZLarge*	%
10	200	474735	100	474721	100	776062	163	564033	119	252314	53	257133	54	252301	53
20	200	462169	100	462141	100	756562	164	547480	118	162464	35	164872	36	162440	35
30	200	459144	100	459103	100	752199	164	543400	118	130646	28	132423	29	130612	28
40	200	462362	100	462307	100	751339	162	548550	119	112734	24	114076	25	112688	24
50	200	460136	100	460067	100	749059	163	544539	118	105467	23	106648	23	105411	23
100	200	468462	100	468324	100	760692	162	556814	119	91291	19	92190	20	91180	19
150	200	454957	100	454750	100	748904	165	540316	119	89329	20	90310	20	89164	20
200	200	452920	100	452640	100	744229	164	535347	118	86911	19	87836	19	86693	19
250	200	428493	100	428157	100	710854	166	507161	118	83595	20	84498	20	83322	19

Abb. 3. Zugriffe auf Text von Erbinformationen; $Tl = 997642$; 1800 Muster

		Skiptest: Zahl der Vergleiche													
Pl	*occurs*	*BMOrig*	%	*BMLarge*	%	*Sunday*	%	*SmithO*	%	*BMZm*	%	*BMZUr1*	%	*BMZLarge*	%
2	175	1275745	100	1344402	105	908696	71	1592889	125	1183757	93	1190341	93	1192513	93
3	763	854667	100	898743	105	679749	80	1188534	139	795399	93	799423	93	803216	84
4	2188	668571	100	710102	106	570702	85	964557	144	609755	91	613588	92	617570	92
5	3169	546617	100	580966	106	489800	90	810668	148	496154	91	499825	91	503651	92
6	3864	472671	100	505222	107	439161	93	704586	149	422753	89	426466	90	430386	91
7	4060	413718	100	442666	107	392977	95	620269	150	367543	89	370630	90	373978	90
8	3614	370568	100	397232	107	357126	96	555218	150	326278	88	329186	89	332371	90
9	3088	337552	100	362559	107	331060	98	504474	149	294501	87	297303	88	300423	89
10	1971	310114	100	333646	108	309128	100	463480	149	268564	87	271228	87	274221	88
11	1119	289005	100	311366	108	290035	100	428746	148	247730	86	250205	87	253035	88
12	592	265242	100	284342	107	266602	101	396274	149	228264	86	230479	87	233020	88
13	280	251973	100	271182	108	255835	102	372700	148	213856	85	216038	86	218584	87
14	116	236872	100	255228	108	242233	102	349582	148	199539	84	201430	85	203586	86
15	44	220123	100	234776	107	224385	102	327749	149	187458	85	189140	86	191138	87
16	17	209609	100	225447	108	214821	102	309278	148	175705	84	177137	85	178756	85
17	7	202323	100	214766	106	207256	102	299214	148	170225	84	171762	85	171033	85
18	4	188173	100	199340	106	192057	102	279035	148	158709	84	159583	85	158080	84
20	1	164609	100	168841	103	160155	97	250844	152	143427	87	143967	87	142331	86
21	2	166050	100	174625	105	163956	99	247102	149	137809	83	138641	83	137440	83
22	1	137663	100	137811	100	131595	96	221827	161	126294	92	126296	92	124265	90

Abb. 4. Vergleiche Muster–Text natürlicher Sprache; $Tl = 2227990$; 25075 Muster

In unseren Tabellen erscheint nur ein Teil der angegebenen Meßwerte. Wie schon für Boyer und Moore sind für uns die Textzugriffe von entscheidendem Gewicht.

1. Der Text wird im allgemeinen so groß sein, daß er nicht in schnellem Cache-Speicher Platz hat.
2. Die meisten Textzugriffe induzieren Feldzugriffe auf die Tabellen der verschiedenen Funktionen. Diese Tabellen könnten durchaus im Cache-Speicher liegen.

Auch die Zahl der Feldzugriffe geht wesentlich in die Laufzeit ein, da jedem Speicherzugriff eine Adreßberechnung vorausgeht.

Sunday hat in [9] die Zahl der Schlüsselvergleiche als entscheidendes Maß angesehen und in [6] moniert, daß die Indexvergleiche von Boyer und Moore verdeckte Schlüsselvergleiche seien. Seinen Vorschlägen entsprechend, sehen wir deshalb alle Vergleiche als gleichwertig an und haben sie in unserer Auswertung entsprechend berücksichtigt.

Eine Tabelle (Abb.6-7) gibt die Fortschrittsgeschwindigkeit unserer Algorithmen bei unterschiedlichen Skip-Funktionen wieder. Wir haben sie relativ zur Zahl der x (x = Textzugriffe, Vergleiche) berechnet, und zwar als (überstrichene Zahl von Zeichen)/x, d.h. als Textlänge/x. Die Prozentzahlen hinter den Tabellenwerten beziehen sich auf den Algorithmus *BMOrig*, dessen Zählerstände zu 100% angesetzt sind.

		Skiptest: Zahl der Vergleiche													
Pl	*occurs*	*BMOrig*	%	*BMLarge*	%	*Sunday*	%	*SmithO*	%	*BMZm*	%	*BMZUr1*	%	*BMZLarge*	%
10	200	700870	100	877581	125	751741	107	688075	98	277705	40	299274	43	327930	47
20	200	683150	100	855948	125	734250	107	668160	98	178251	26	191123	28	209597	31
30	200	680419	100	853336	125	732862	108	664657	98	143492	21	153658	23	168591	25
40	200	682200	100	854127	125	727662	107	668672	98	123621	18	132052	19	144799	21
50	200	678717	100	848976	125	725554	107	663994	98	115676	17	123626	18	135873	20
100	200	692854	100	867880	125	738283	107	679498	98	100033	14	106723	15	117193	17
150	200	672933	100	841951	125	727616	108	660036	98	98163	15	104872	16	115028	17
200	200	671950	100	842177	125	726318	108	655063	97	95651	14	102131	15	111890	17
250	200	632699	100	791137	125	689213	109	618768	98	91897	15	98067	15	107165	17

Abb. 5. Vergleiche Muster–Text von Erbinformationen; $Tl = 997642$; 1800 Muster

		Skiptest: Durchschnittliche Verschiebung pro Textzugriff													
Pl	*occurs*	*BMOrig*	%	*BMLarge*	%	*Sunday*	%	*SmithO*	%	*BMZm*	%	*BMZUr1*	%	*BMZLarge*	%
10	200	2.10	100	2.10	100	1.29	61	1.77	84	3.95	188	3.88	185	3.95	188
20	200	2.16	100	2.16	100	1.32	61	1.82	84	6.14	284	6.05	280	6.14	284
30	200	2.17	100	2.17	100	1.33	61	1.84	85	7.64	352	7.53	347	7.64	352
40	200	2.16	100	2.16	100	1.33	62	1.82	84	8.85	410	8.75	405	8.85	410
50	200	2.17	100	2.17	100	1.33	61	1.83	84	9.46	436	9.35	431	9.46	436
100	200	2.13	100	2.13	100	1.31	62	1.79	84	10.93	513	10.82	508	10.94	514
150	200	2.19	100	2.19	100	1.33	61	1.85	84	11.17	510	11.05	505	11.19	511
200	200	2.20	100	2.20	100	1.34	61	1.86	85	11.48	522	11.36	516	11.51	523
250	200	2.33	100	2.33	100	1.40	60	1.97	85	11.93	512	11.81	507	11.97	514

Abb. 6. *Tl*/(Z.d. Textzugriffe) Erbinformationen; $Tl = 997642$; 1800 Muster

		Skiptest: Durchschnittliche Verschiebung pro Vergleich													
Pl	*occurs*	*BMOrig*	%	*BMLarge*	%	*Sunday*	%	*SmithO*	%	*BMZm*	%	*BMZUr1*	%	*BMZLarge*	%
10	200	1.42	100	1.14	80	1.33	94	1.45	102	3.59	253	3.33	235	3.04	214
20	200	1.46	100	1.17	80	1.36	93	1.49	102	5.60	384	5.22	357	4.76	326
30	200	1.47	100	1.17	80	1.36	93	1.50	102	6.95	473	6.49	441	5.92	403
40	200	1.46	100	1.17	80	1.37	94	1.49	102	8.07	553	7.55	517	6.89	472
50	200	1.47	100	1.18	80	1.38	94	1.50	102	8.62	586	8.07	549	7.34	500
100	200	1.44	100	1.15	80	1.35	94	1.47	102	9.97	692	9.35	649	8.51	591
150	200	1.48	100	1.18	79	1.37	93	1.51	102	10.16	686	9.51	643	8.67	586
200	200	1.48	100	1.18	80	1.37	93	1.52	103	10.43	705	9.77	660	8.92	603
250	200	1.58	100	1.26	80	1.45	92	1.61	102	10.86	687	10.17	644	9.31	589

Abb. 7. *Tl*/(Z. d. Vergleiche) Erbinformationen; $Tl = 997642$; 1800 Muster

		Shifttest: Zahl der Textzugriffe													
Pl	*occurs*	*BMO*	%	*BMH*	%	*BM1D*	%	*BM2D*	%	*BMSZ*	%	*BMHZ*	%	*BMZ*	%
2	175	1202269	100	1170733	97	1170733	97	1202269	100	1170733	97	1168458	97	1168458	97
3	200	813816	100	792374	97	792236	97	813384	100	792374	97	789946	97	790055	97
4	200	629764	100	609138	97	608975	97	629532	100	609077	97	605676	96	605831	96
5	200	511663	100	494944	97	494898	97	511503	100	494902	97	491276	96	491435	96
6	200	420930	100	406023	96	405785	96	420450	100	405881	96	413953	98	414108	98
7	200	381980	100	368956	97	368645	97	381397	100	368685	97	362306	95	362455	95
8	200	342631	100	330930	97	330413	96	342044	100	330554	96	322654	94	322817	94
9	200	309304	100	298835	97	298391	96	308615	100	298434	96	289923	94	290084	94
10	200	283375	100	273963	97	273271	96	282501	100	273272	96	263917	93	264062	93
11	200	261977	100	253633	97	252836	97	261172	100	252996	97	243005	93	243134	93
12	200	243674	100	235504	97	234857	96	242745	100	234824	96	224756	92	224895	92
13	200	237490	100	230666	97	229604	97	236438	100	229752	97	218192	92	218303	92
14	116	222834	100	216186	97	215505	97	221872	100	215335	97	203841	91	203980	92
15	44	210051	100	205229	98	204389	97	208708	99	204133	97	191336	91	191411	91

Abb. 8. Zugriffe auf Text natürlicher Sprache; $Tl = 2227990$; 2535 Muster

		Shifttest: Zahl der Textzugriffe													
Pl	*occurs*	*BMO*	%	*BMH*	%	*BM1D*	%	*BM2D*	%	*BMSZ*	%	*BMHZ*	%	*BMZ*	%
10	200	362928	100	381603	105	319379	88	337383	93	335921	93	227724	63	231802	64
20	200	306333	100	370724	121	268865	88	257081	84	278361	91	146541	48	146482	48
30	200	274798	100	361081	131	240901	88	215754	79	257348	94	119429	43	116754	42
40	200	259800	100	354342	136	227148	87	192682	74	244731	94	103552	40	99584	38
50	200	250848	100	357280	142	219861	88	182915	73	241340	96	96684	39	91754	37
100	200	226260	100	374438	165	197549	87	149256	66	255263	113	83950	37	75891	34
150	200	207473	100	369422	178	180447	87	129610	62	241658	116	81671	39	71797	35
200	200	191503	100	362595	189	166714	87	118916	62	235511	123	79998	42	70408	37
250	200	186459	100	348378	187	163649	88	110612	59	235834	126	77899	42	68864	37

Abb. 9. Zugriffe auf Text v. Erbinformationen; $Tl = 997642$; 1800 Muster

		Shifttest: Zahl der Vergleiche													
Pl	*occurs*	*BMO*	%	*BMH*	%	*BM1D*	%	*BM2D*	%	*BMSZ*	%	*BMHZ*	%	*BMZ*	%
2	175	1256304	100	1224768	97	1224768	97	1224768	97	1193232	95	1181958	94	1181958	94
3	200	839563	100	818134	97	818000	97	817145	97	796165	95	790895	94	788683	94
4	200	654690	100	634123	97	633962	97	633164	97	612744	94	606311	93	603661	92
5	200	531675	100	514965	97	514922	97	514490	97	497893	94	491696	92	489460	92
6	200	455789	100	440964	97	440689	97	439465	96	424940	93	416971	91	414615	91
7	200	399459	100	386528	97	386209	97	384370	96	371670	93	362895	91	360934	90
8	200	359111	100	347576	97	347003	97	345132	96	333651	93	323266	90	321267	89
9	200	324556	100	314212	97	313737	97	311689	96	301518	93	290617	90	288754	89
10	200	298175	100	288952	97	288202	97	285552	96	276334	93	264666	89	262926	88
11	200	275654	100	267506	97	266655	97	264359	96	256195	93	243941	88	242305	88
12	200	256981	100	248982	97	248278	97	245913	96	237989	93	225721	88	224171	87
13	200	249972	100	243354	97	242209	97	239467	96	232767	93	219085	88	217624	87
14	116	234431	100	228028	97	227288	97	224464	96	217939	93	204569	87	203348	87
15	44	219931	100	215276	98	214411	97	211251	96	206689	94	192145	87	190987	87

Abb. 10. Vergleiche Muster–Text natürlicher Sprache; $Tl = 2227990$; 2535 Muster

		Shifttest: Zahl der Vergleiche													
Pl	*occurs*	*BMO*	%	*BMH*	%	*BM1D*	%	*BM2D*	%	*BMSZ*	%	*BMHZ*	%	*BMZ*	%
10	200	450073	100	494665	110	410378	91	368131	82	373761	83	235984	52	225483	50
20	200	379197	100	481102	127	344813	91	279927	74	309634	82	151639	40	142580	38
30	200	340532	100	469623	138	309287	91	235079	69	286458	84	123775	36	113787	33
40	200	320911	100	459265	143	290513	91	209360	65	272049	85	107277	33	97148	30
50	200	309839	100	463375	150	281196	91	198809	64	268616	87	100093	32	89467	29
100	200	279195	100	486067	174	252336	90	161920	58	283902	102	86842	31	74035	27
150	200	256035	100	480041	187	230411	90	140783	55	269085	105	84697	33	70189	27
200	200	236742	100	472295	199	213405	90	129266	55	262495	111	83145	35	69027	29
250	200	230099	100	452299	197	209031	91	120178	52	262617	114	80920	35	67502	29

Abb. 11. Vergleiche Muster–Text v. Erbinformationen; $Tl = 997642$; 1800 Muster

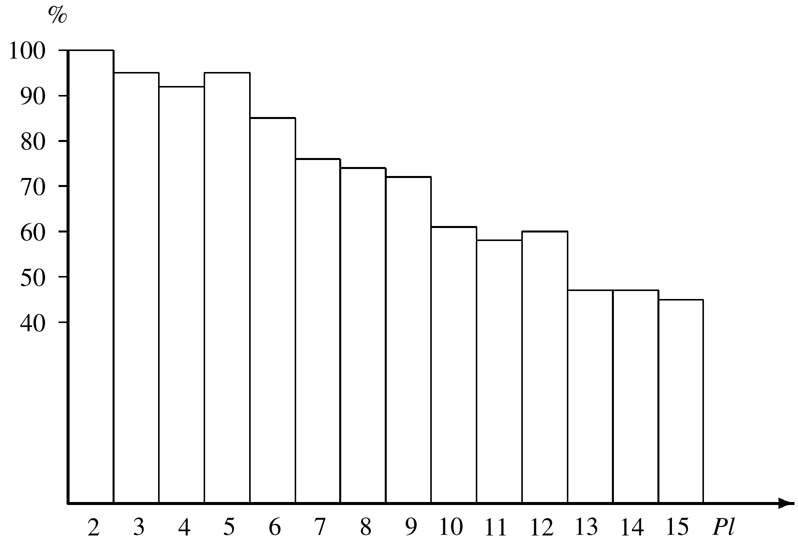

Abb. 12. $(CShift \geq Pl - 1)$ -v.H.-Anteil bei natürlicher Sprache; 25075 Muster