SOFTWARE—PRACTICE AND EXPERIENCE, VOL. 21(10), 1065–1074 (OCTOBER 1991)

Experiments with a Very Fast Substring Search Algorithm

P. D. SMITH
Department of Computer Science, California State University, 18111 Nordhoff St. COMS, Northridge, CA 91330, U.S.A.

SUMMARY

Sunday devised string matching methods that are generally faster than the Boyer-Moore algorithm. His fastest method used statistics of the language being scanned to determine the order in which character pairs are to be compared. In this paper the performances of similar, but language-independent, algorithms are examined. Results comparable with language-based algorithms can he achieved with an adaptive technique. In terms of character comparisons, a faster algorithm than Sunday's is constructed by using the larger of two pattern shifts.

KEY WORDS Pattern matching String matching Searching Adaptive Sunday

BACKGROUND

A string matching problem is to find the occurrences of a *pattern* of M characters ($P = P_1, \ldots, P_M$) in a generally very much longer text string T ($T_1, \ldots, T_N$). The solution has applications in editors and information retrieval programs. Notable algorithms to perform the search have been devised by Knuth, Morris and Pratt [1] and by Boyer and Moore. [2]

During a search, the pattern may be considered aligned with part of the text (see Figure 1). Algorithms vary in the order in which character-character comparisons are made and in the way that they determine the distance the pattern can be shifted if the current alignment does not match.

In the straightforward method, corresponding pairs of characters are compared left-to-right. If a mismatch is found then the pattern moves one place down the text, i.e. so that P_1 is aligned with T_{k+1}. In the Boyer–Moore algorithm, character-pairs are compared right-to-left, that is the first comparison is between P_M and T_{k+M-1}. If a comparison fails, e.g. P_j with T_{k+j-1}, then both the index of the failure point (j) and the identity of the text character (T_{k+j-1}) are used to determine the distance

Text	...	T_k	T_{k+1}	...	T_{k+j-1}	...	T_{k+M-1}	T_{k+M}	...
Pattern		P_1	P_2	...	P_j	...	P_M		

Figure 1. Pattern–text alignment

0038–0644/91/101065–10$05.00

Received 28 November 1990
Revised 16 April 1991

that the pattern can be moved down the text for the next potential match. Two tables of distances are computed from P before searching starts: table delta1 is indexed by T_{k+j-1} and table delta2 is indexed by j.

SUNDAY'S ALGORITHM

Sunday's [3] insight into the string matching problem was to observe that if the pattern does not match the currently aligned text fragment, the character T_{k+M} must be aligned with the pattern in its next position unless the pattern is shifted right past it. He uses this character to determine the shift amount, again using a table (DELTA1) precomputed from P. If T_{k+M} is in P then it is aligned with the rightmost instance. Using T_{k+M} to determine the shift distance has two consequences. First, if the charcter is not in P we can shift the pattern right past it. Thus, the greatest distance we can shift the pattern is one greater than the greatest shift possible with Boyer–Moore. Secondly, we are free to compare character pairs in any order. In particular, we can test first the pair that gives the best shift if there is a mismatch or test first the pair that seems most likely to be different. For example, we can compare characters in the pattern in ascending order of their *a priori* frequency in the language. Sunday termed this approach the optimal mismatch (OM) technique. His experiments showed that it results in fewer character comparisons than the Boyer–Moore algorithm (though the percentage improvement declines with increasing pattern length).

There are two aspects of a fast string searching algorithm. The first is the quick detection of a mismatch between the pattern and the aligned text fragment; checking in character frequency order is effective here. The second is the largest possible shift of the pattern. In this paper we first examine the influence of the ordering of character pairs on mismatch detection and then present a variation on OM having a larger average shift.

MISMATCH DETECTION

How can OM be used if the frequency distribution is not known? A strategy in this case is to start with some arbitrary ordering and allow it to adapt as searching progresses. In our experiments, a position in the pattern that results in a mismatch is moved to the front of the ordering list so that in the next alignment, that position is tried first. The other positions are thus shifted down one place. An alternative is to move the mismatch position up one place in the list. Figure 2 gives pseudocode for the matching of a text fragment with the pattern. If the text matches the pattern

```
i ← l
while i ≤ patternlength
   do if Pattern[Ordering[i]] = Text[offset+ Ordering[i]]
      then i ← i +l
      else { if i> 1 then Adjust (Ordering,i)
             i ← patternlength + 2
           }
```

Figure 2. Pattern–text comparison

the final value of *i* is pattern length+1, otherwise it is pattern length+2. The ordering is static if the call to Adjust is omitted.

We compare the performance of such an adaptive algorithm with algorithms that use static language-based orderings.

EXPERIMENTS WITH ORDERINGS

Sunday's OM shifts by the larger of two quantities—the DELTA1 noted above and a DELTA2 similar to the Boyer–Moore delta2. The latter depends on the order in which characters are matched; hence it has to be recomputed whenever the ordering is modified. Because of this large overhead DELTA2 is omitted from the adaptive algorithms. In order to make valid comparisons it is also omitted it from the version of OM used here. The resulting algorithm is designated OM-1. Note that without DELTA2 there is no linear bound on worst-case behaviour and the search may be slower for large *M* where DELTA2 has greater effect.

In our implementation of Sunday's algorithm, four different initial orderings of the pattern positions were allowed:

(i) ascending order of frequency in English
(ii) ascending order of frequency in the programming language C [4]
(iii) right-to-left
(iv) random.

In addition, the ordering could be either static or adaptive (as described above). Thus there were eight variations of the basic algorithm. The C option was chosen because editing C source (including string searching) is a common activity in our local environment. The right-to-left option gives an algorithm similar to Boyer–Moore. In addition, two other trials were conducted: one using the frequency distribution of the text being searched (the 'optimal' distribution) and the second using the inverse of this distribution (the 'pessimal' distribution).

What are termed 'generic' frequency distributions for English and C were obtained. Data for English character frequencies was derived from a novel; [5] data for character frequencies in C was derived from the source of the Gnu C compiler. In both cases relative frequencies were stored to three decimal places (thus characters occurring less than once per 2000 characters were regarded as never occurring).

The text files used to measure the efficiency of substring searching were different from those used to obtain frequency data. For English the file was a short story; [6] for C the file was the source code of a message handling shell. [7] In addition, a file of French text was used, made up of contributions to the soc.culture.french Usenet newsgroup. These particular files were used because of their similar sizes (53,452, 53,478 and 53,523 bytes respectively), file length being significant in the event of an unsuccessful search. The character frequency data for French is shown with the English and C generic data in Table I (upper case letters omitted).

Our experimental method was similar to that of Smit. [8] From each of the three text files, 18 strings of each length from 1 to 14 characters were selected at random. In addition, for each of the lengths, two strings were generated that were not in the file. For a particular text file, the first occurrences of each of the appropriate 280 strings were searched for using each of the 10 variations of the algorithm.

Table I. Character frequency data

Character	Percentage of English	C	French	Character	Percentage of English	C	French
tab	0·0	1·3	0·2	newline	2·2	3·8	2·1
space	16·3	21·3	15·6	!	0·1	0·2	0·1
"	0·3	0·3	0·3	#	0·0	0·2	0·0
&	0·0	0·3	0·0	'	0·8	0·7	1·0
(	0·0	1·1	0·1	)	0·0	1·1	0·1
*	0·0	1·5	0·0	+	0·1	0·6	0·0
,	1·3	0·8	0·9	−	0·5	0·8	0·3
.	0·9	0·6	1·1	/	0·0	0·9	0·0
0	0·0	0·3	0·0	1	0·0	0·4	0·1
2	0·0	0·2	0·0	:	0·0	0·2	0·2
;	0·1	1·5	0·0	<	0·1	0·1	0·0
=	0·0	1·2	0·0	>	0·1	0·5	0·0
?	0·1	0·0	0·1	[	0·0	0·3	0·0
\	0·0	0·2	0·0	]	0·0	0·3	0·0
_	0·0	1·2	0·0	‘	0·0	0·0	0·1
a	6·2	3·6	6·1	b	1·2	1·3	0·7
c	1·7	2·1	2·4	d	3·5	1·8	2·7
e	9·4	6·3	13·0	f	1·6	2·3	0·8
g	1·6	1·0	0·6	h	4·9	1·7	0·5
i	4·8	4·6	5·5	j	0·1	0·0	0·3
k	0·6	0·5	0·0	i	3·0	2·1	4·1
m	1·8	1·3	2·1	n	5·3	4·0	5·7
o	5·8	2·9	4·2	p	1·2	2·5	2·2
q	0.1	0·0	0·9	r	4·3	3·3	5·1
s	4·8	3·1	6·2	t	6·5	4·5	5·3
u	2·1	1·6	4·6	v	0·7	0·3	0·9
w	1·8	0·7	0·0	x	0·1	0·3	0·3
y	1·6	0·4	0·3	z	0·0	0·2	0·1
{	0·0	0·3	0·0	|	0·0	0·1	0·0
}	0·0	0·3	0·0	~	0·0	0·0	0·2

RESULTS

Table II shows the results of testing the algorithms on the English source. A measure of efficiency of a searching method is the proportion of text characters preceding the place where the pattern is found that are compared with pattern characters. This is the measure used by Smit and is the one reported here. Each entry in the table is the average value (over 20 searches) of:

$$R = \frac{\text{number of character–character comparisons}}{\text{number of characters passed in the text string}}$$

Note that the alignments of a particular pattern against a particular text are the same for each of the 10 variations. Therefore differences in search efficiency are due entirely to the differences in the number of comparisons required to detect a mismatch.

Allowing the two language-based orderings to adapt during searching made little

Table II. Trials on English text

	Initial generic English		Initial generic C		Initial R → L		Initial random		Optimal	Pessimal
	(a)	(b)	(c)	(d)	(e)	(f)	(g)	(h)	English	English
M	Static	Dynamic	Static	Dynamic	Static	Dynamic	Static	Dynamic	(static)	(static)
1	0·604	0·604	0·604	0·604	0·604	0·604	0·604	0·604	0·604	0·604
2	0·385	0·388	0·387	0·389	0·431	0·390	0·431	0·390	0·378	0448
3	0·326	0·331	0·328	0·332	0·390	0·335	0·390	0·335	0·313	0·390
4	0·241	0·243	0·242	0·243	0·264	0·245	0·262	0·246	0·227	0·271
5	0·215	0·218	0·217	0·219	0·236	0·220	0·251	0·220	0·197	0·258
6	0·199	0·199	0·199	0·199	0·244	0·204	0·220	0·203	0·181	0·248
7	0·173	0·175	0·174	0·175	0·201	0·178	0·194	0·181	0·154	0·201
8	0·154	0·155	0·156	0·155	0·178	0·161	0·178	0·162	0·134	0·175
9	0·137	0·138	0·137	0·138	0·150	0·140	0·152	0·140	0·117	0·151
10	0·135	0·136	0·136	0·136	0·159	0·140	0·158	0·141	0·111	0·142
11	0·124	0·125	0·125	0·125	0·145	0·129	0·133	0·129	0·105	0·138
12	0·121	0·121	0·121	0·122	0·142	0·127	0·141	0·126	0·102	0·142
13	0·114	0·115	0·115	0·115	0·128	0·120	0·127	0·119	0096	0·125
14	0·109	0·110	0·110	0·110	0·122	0·114	0·122	0·115	0·089	0·117

difference (compare column (a) with column (b) and column (c) with column (d)). If anything, it slowed searching. This is probably due to a good ordering being disrupted by the crude move-to-front strategy used to organize the list. Adaptive language-independent orderings performed well, though not quite as well as static language-based searches.

Table III shows the first eight columns of Table I transformed to a scale where the performance of the optimal distribution is 0 and that of the pessimal distribution

Table III. Trials on English text—scaled results

	Initial generic English		Initial generic C		Initial R → L		Initial random	
	(a)	(b)	(c)	(d)	(e)	(f)	(g)	(h)
M	Static	Dynamic	Static	Dynamic	Static	Dynamic	Static	Dynamic
1	–	–	–	–	–	–	–	–
2	10·0	14·3	12·9	15·7	75·7	17·1	75·7	17·1
3	16·9	23·4	19·5	24·7	100·0	28·6	100·0	28·6
4	31·8	36·4	34·1	36·4	84·1	40·9	79·5	43·2
5	29·5	34·4	32·8	36·1	63·9	37·7	88·5	37·7
6	26·9	26·9	26·9	26·9	94·0	34·3	58·2	32·8
7	40·4	44·7	42·6	44·7	100·0	51·1	85·1	57·4
8	48·8	51·2	53·7	51·2	107·3	65·9	107·3	68·3
9	58·8	61·8	58·8	61·8	97·1	67·6	102·9	67·6
10	77·4	80·6	80·6	80·6	154·8	93·5	151·6	96·8
11	57·6	60·6	60·6	60·6	121·2	72·7	84·8	72·7
12	47·5	47·5	47·5	50·0	100·0	62·5	97·5	60·0
13	62·1	65·5	65·5	65·5	110·3	82·8	106·9	79·3
14	71·4	75·0	75·0	75·0	117·9	89·3	117·9	92·9

is 100. Using generic information gives surprisingly poor performance for larger *M*. Using a dynamic language-independent ordering rather than a language-based ordering gives performances about 1·5 times further away from the optimal.

One surprising aspect of Table II is how well the C-based orderings performed on English text. This may be because good C source code contains many comments and meaningful identifiers, and hence the rankings of characters by frequency in our samples of C and English are similar.

Table IV shows the results of testing the algorithms on the C text. Again, the two language-based methods were almost equally as good. In contrast to Table II, allowing language-based orderings to adapt often led to a slightly faster search. Also, adaptive language-independent orderings produced searches very nearly as fast as language-based methods. Table V shows the results of Table IV scaled against optimal and pessimal C, and it is apparent that dynamic language-independent orderings are very competitive with generic language data. Some of the overall differences between Tables II and IV might be due to the larger character set used by C. In our samples, C used 74 characters and English used 55 characters.

The third set of tests used the French text. They were designed to evaluate the algorithms on a 'foreign' language. Table VI shows the results. Surprisingly, the language-based methods searched French more efficiently than English.

PATTERN SHIFTING

Sunday's OM was also evaluated in trials similar to those he reported. [3] All words of each length from 1 to 15 were extracted from a copy of the lexicon used by the Unix* spell utility in which upper case letters had been converted to lower case. All occurrences of each word were then searched for in the modified lexicon. It seemed

Table IV. Trials on C text

	Initial generic English		Initial generic C		Initial R → L		Initial random		Optimal	Pessimal
	(a)	(b)	(c)	(d)	(e)	(f)	(g)	(h)	C	C
M	Static	Dynamic	Static	Dynamic	Static	Dynamic	Static	Dynamic	(static)	(static)
1	0·583	0·583	0·583	0·583	0·583	0·583	0·583	0·583	0·583	0·583
2	0·368	0·370	0·368	0·370	0·384	0·376	0·384	0·376	0·361	0·374
3	0·286	0·283	0·285	0·283	0·294	0·285	0·294	0·285	0·272	0·296
4	0·230	0·229	0·229	0·228	0·244	0·229	0·241	0·229	0·218	0·243
5	0·195	0·194	0·195	0·194	0·214	0·196	0·207	0·196	0·183	0·214
6	0·170	0·169	0·170	0·170	0·187	0·170	0·182	0·170	0·159	0·194
7	0·153	0·151	0·152	0·151	0·166	0·153	0·160	0·154	0·140	0·177
8	0·150	0·142	0·149	0·142	0·163	0·145	0·151	0·144	0·133	0·167
9	0·127	0·126	0·126	0·125	0·136	0·127	0·137	0·128	0·114	0·151
10	0·117	0·115	0·116	0·115	0·125	0·117	0·125	0·117	0·103	0·129
11	0·108	0·108	0·107	0·107	0·114	0·109	0·114	0·108	0·097	0·120
12	0·106	0·103	0·104	0·102	0·111	0·104	0·107	0·104	0·094	0·120
13	0·095	0·093	0·093	0·093	0·100	0·094	0·101	0·093	0·082	0·107
14	0·092	0·091	0·091	0·091	0·101	0·092	0·098	0·093	0·080	0·105

* Unix is a trademark of AT&T.

Table V. Trials on C text—scaled results

	Initial generic English		Initial generic C		Initial R → L		Initial random	
	(a)	(b)	(c)	(d)	(e)	(f)	(g)	(h)
M	Static	Dynamic	Static	Dynamic	Static	Dynamic	Static	Dynamic
1	-	-	-	-	-	-	-	-
2	53·8	69·2	53·8	69·2	176·9	115·4	176·9	115·4
3	58·3	45·8	54·2	45·8	91·7	54·2	91·7	54·2
4	48·0	44·0	44·0	40·0	104·0	44·0	92·0	44·0
5	38·7	35·5	38·7	35·5	100·0	41·9	77·4	41·9
6	31·4	28·6	31·4	31·4	80·0	31·4	65·7	31·4
7	35·1	29·7	32·4	29·7	70·3	35·1	54·1	37·8
8	50·0	26·5	47·1	26·5	88·2	35·3	52·9	32·4
9	35·1	32·4	32·4	29·7	59·5	35·1	62·2	37·8
10	53·8	46·2	50·0	46·2	84·6	53·8	84·6	53·8
11	47·8	47·8	43·5	43·5	73·9	52·2	73·9	47·8
12	46·2	34·6	38·5	30·8	65·4	38·5	50·0	38·5
13	52·0	44·0	44·0	44·0	72·0	48·0	76·0	44·0
14	48·0	44·0	44·0	44·0	84·0	48·0	72·0	52·0

Table VI. Trials on French text

	Initial English		Initial C		Initial R → L		Initial random	
	(a)	(b)	(c)	(d)	(e)	(f)	(g)	(h)
M	Static	Dynamic	Static	Dynamic	Static	Dynamic	Static	Dynamic
1	0·631	0·631	0·631	0·631	0·631	0·631	0·631	0·631
2	0·406	0·409	0·406	0·409	0·435	0·411	0·435	0·411
3	0·325	0·330	0·325	0·332	0·354	0·338	0·354	0·338
4	0·242	0·243	0·242	0·243	0·268	0·245	0·266	0·245
5	0·209	0·211	0·210	0·211	0·230	0·214	0·225	0·215
6	0·186	0·188	0·187	0·188	0·204	0·192	0·201	0·191
7	0·163	0·164	0·164	0·164	0·175	0·168	0·177	0·167
8	0·150	0·151	0·151	0·152	0·167	0·155	0·166	0·153
9	0·138	0·138	0·138	0·138	0·151	0·144	0·154	0·141
10	0·133	0·132	0·132	0·133	0·147	0·138	0·146	0·140
11	0·128	0·128	0·129	0·129	0·139	0·134	0·137	0·133
12	0·123	0·123	0·124	0·123	0·137	0·129	0·141	0·131
13	0·109	0·110	0·110	0·110	0·125	0·115	0·124	0·114
14	0·104	0·104	0-104	0·105	0·115	0·108	0·112	0·108

likely that in these particular trials, DELTA2 would not be generally helpful—the patterns are small and the alphabet is comparatively large· Table VII summarizes the efficiencies of the searches using the same metric as the earlier tables and it can be seen that OM-1 performed nearly as well as OM.

There are cases where taking the larger of two DELTA1 values is beneficial. Consider the example of Figure 3.

Algorithm OM2 uses the following pattern shift:

Table VII. Trials on an English lexicon

M	OM-1	OM	OM2	OM3	Minimum	OM2 as a percentage of OM	OM3 as a percentage of OM2
1	0·509	0·509	0·509	0·509	0·500	100·0	100·0
2	0·371	0·370	0·361	0·361	0·333	97·6	100·O
3	0·285	0·284	0·272	0·271	0·250	95·8	99·6
4	0·234	0·233	0·219	0·218	0·200	94·0	99·5
5	0·200	0·200	0·184	0·182	0·167	92·0	98·9
6	0·177	0·177	0·159	0·157	0·143	89·8	98·7
7	0·158	0·158	0·141	0·137	0·125	89·2	97·2
8	0·144	0·144	0·126	0·123	0·111	87·5	97·6
9	0·134	0·133	0·115	0·111	0·100	86·5	96·5
10	0·125	0·125	0·106	0·102	0·091	84·8	96·2
11	0·119	0·118	0·099	0·094	0·083	83·9	94·9
12	0·112	0·111	0·093	0·088	0·077	83·8	94·6
13	0·107	0·107	0·088	0·082	0·071	82·2	93·2
14	0·102	0·101	0·083	0·077	04367	82·2	92·8
15	0·099	0·098	0·079	0·073	0·063	80·6	92·4

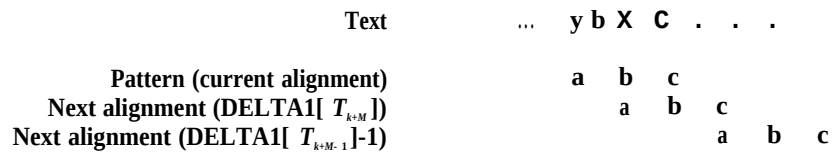

Figure 3. Shift differences

$$\max (\mathrm{DELTA1} [T_{k+M}], \mathrm{DELTA1} [T_{k+M-1}]\text{-}1)$$

Although the second component has a maximum value one less than the first, column 3 of Table VII shows that using OM2 results in an efficient algorithm.

We can take the idea further and use OM3, where the pattern shift is

$$\max (\mathrm{DELTA1} [T_{k+M}], \mathrm{DELTA1} [T_{k+M-1}]\text{–l}, \mathrm{DELTA1} [T_{k+M-2}]\text{-}2)$$

The results of using this are shown in column 4. However, the final column shows that improvements are diminishing as the number of parameters of max is increased.

If a pattern–text mismatch is detected with just one comparison, and shifts the pattern the maximum distance, the expected value of the ratio R will be $1/(M+1)$. This lower bound is shown in column 5. Some of the data of Table VII are presented graphically in Figure 4. The longer the pattern, the greater is the improvement in performance of OM2 over OM. Table VIII shows why this is so; it reports the percentages with which each of the three relations between the two shift quantities occurred. The longer the pattern, the more likely it is that a case like Figure 3 arises.

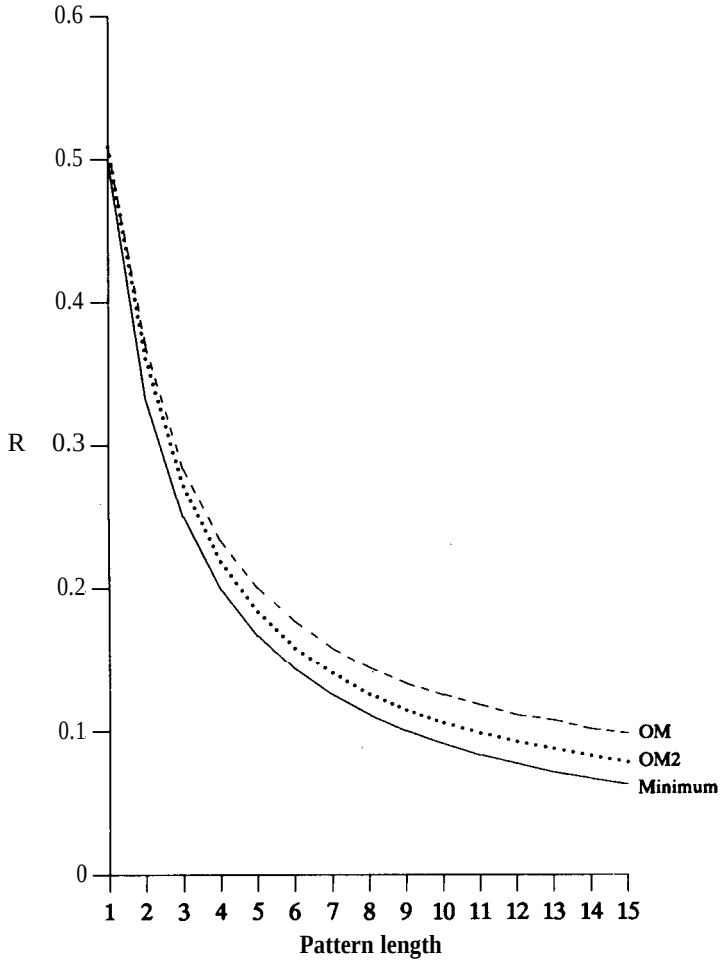

Figure 4. Trials on an English

CONCLUSIONS

OM can be made generic. On English, French and C text, starting with either a right-to-left or a random ordering and allowing it to adapt can yield performances comparable with those achieved with language-based orderings. Thus a generic editor, for example, need not be told the type of text it is searching in order to search it efficiently.

Greater shifts are achieved when the largest of a set of shift values is used instead of any one shift exclusively. The maxima can be precomputed from the pattern and stored in a table. If the alphabet contains 2^d characters then OM k needs a table with 2^{kd} entries. For example, OM2 operating on ASCII needs a table of 16K shifts.

Table VIII. Distribution of shift cases

M	DELTA1 [T_{k+M}] larger	Equal	DELTA1 [T_{k+M-1}] - 1
1	96·8	3·2	0·0
2	91·3	4·3	4·4
3	87·1	3·9	9·0
4	83·1	3·7	13·1
5	79·3	4·0	16·7
6	76·0	4·2	19·8
7	73·5	4·2	22·4
8	71·0	4·3	24·8
9	68·5	4·3	27·2
10	66·4	4·5	29·2
11	64·6	4·6	30·7
12	63·5	4·6	31·9
13	62·3	4·6	33·1
14	61·4	4·5	34·1
15	60·3	4·8	34·9

ACKNOWLEDGEMENTS

Thanks are due to Mike Barnes for comments on drafts of this paper and assistance with analysis of the experimental results and to the referees for constructive criticism and suggestions.

REFERENCES

1. D. E. Knuth, J. H. Morris, Jr and V. B. Pratt, 'Fast pattern matching in strings', *SIAM J. Computing*, **6,** 323–350 (1977).
2. R. S. Boyer and J. S. Moore, 'A fast string search algorithm', *Comm. ACM, 20, 762–772 (1977).*
3. D. M. Sunday, 'A very fast substring search algorithm', *Comm. ACM,* **33,** 132–142 (1990).
4. B. W. Kernighan and D. M. Ritchie, The C *Programming Language,* Prentice-Hall, Englewood Cliffs, NJ, 1978.
5. T. Hardy, *Far from the Madding Crowd, 1874.* Penguin Edition, 1978.
6. A. C. Doyle, 'The Boscombe Valley mystery', in *The Adventures of Sherlock Holmes,* Berkley Publishing Corporation, New York, NY, 1963. First published 1892.
7. M. T. Rose and J. L. Romine, *The Rand MH Message Handling system: User's Manual (UCI Version),* University of California, Irvine, 1989.
8. G. De V. Smit, 'A comparison of three string matching algorithms', *Software—Practice and Experience,* **12,** 57–66 (1982).