NORTH-HOLLAND

Informatics and Computer Science

Experimenting with Pattern-Matching Algorithms

YANNIS MANOLOPOULOS*

Department of Informatics, Aristotle University, Thessaloniki, Greece 54006

and

CHRISTOS FALOUTSOS*

Department of Computer Science, University of Maryland, College Park, Maryland 20742

ABSTRACT

Two new pattern-matching algorithms based on the Boyer-Moore algorithm are presented. Their performance is compared to that of earlier relevant variants in terms of the number of character comparisons and the required running time by exhaustive simulation. Experimental results show the efficiency of both these two new algorithms.

1. INTRODUCTION

Text-searching methods have been divided in three categories: (a) *full text scanning*, (b) *text inversion*, and (c) *signature files* [5]. The first category includes the highly honored research topic of *pattern-matching*. Milestones in this area are the *BM algorithm* by Boyer and Moore [4] and the *KMP algorithm* by Knuth, Morris, and Pratt [8], which appeared in the late 1970s. Since then, many efforts have been reported; [1, 2] give pointers towards this rich literature. The reason that the topic still remains open is its importance in a number of applications, such as in text editors, word processors, lexical analyzers, information retrieval systems, or even in vision for two-dimensional image recognition or in biology for molecular sequence analysis.

*Currently on leave at AT&T Bell Laboratories, Murray Hill.

0020-0255/96/$15.00
SSDI 0020-0255(95)00280-4

New methods have appeared more recently. For example, Sunday reported three new enhancements of the BM algorithm [11]. Then, Smith elaborated on a variant proposed by Sunday [10]. Finally, a synthetic and exhaustive experimentation on a score of BM variants was reported by Hume and Sunday [7]. In the present paper, we experimentally test and compare the efficiency of some known BM variants as well as two new ones. In the next section, we will introduce very briefly some BM variations. This way, we can smoothly present and explain the two new ones in Section 3. Experimental results comparing the algorithms' performance in terms of the number of character comparisons and the required running time are included in Section 4.

2. ALGORITHM PRESENTATION

The simplest *naive* algorithm uses two pointers which are initialized to point to the first pattern character and the first text character. If these characters match, then the pointers are advanced by one position and the comparison of succeeding pattern and text characters continues the apparent way. In case of mismatch, the pattern pointer is initialized, whereas the text pointer is advanced by one position. Thus, it seems as if after a mismatch the pattern slides only one position on top of the text. The authors of [4, 8] recognized the fact that trying to search for a pattern in the text by shifting the pattern only one position after a mismatch is a memoryless procedure, e.g., the information obtained from the matches and the final mismatch is lost.

BOYER-MOORE METHOD

The Boyer-Moore algorithm is based on (a) the *occurrence heuristic*, and (b) the *match heuristic* (terminology according to Baeza-Yates [2]). These heuristic techniques are used to reposition the pattern after a mismatch. We also notice that a basic characteristic of the BM algorithm is that comparisons start from the rightmost pattern character proceeding towards the leftmost ones.

According to the occurrence heuristic, the pattern has to be shifted after a mismatch, so that on top of the text character where the mismatch occurred, a same pattern character will be positioned. For this reason, an *occurrence table* with length equal to the alphabet size is built in a preprocessing phase. Entries of the table corresponding to alphabet characters which do not exist in the pattern are assigned a value equal to the pattern length. Entries corresponding to characters appearing in the pattern are assigned a value equal to the shorter distance between this character and the last one. Thus, the table entry for the last pattern character is equal to 0.

According to the match heuristic, we have to take into consideration any existing subpatterns. Thus, after a mismatch, the pattern should be shifted so that there are matches for all the characters that matched before, and, in addition, a different pattern character is positioned on top of the text character where the mismatch occurred. Again, a *match table* with length equal to that of the pattern is built before actual processing starts. Each table entry takes a value equal to the distance of the specific character x to a different character y such that the characters succeeding x to the pattern end are the same with the characters succeeding y.

Thus, whenever a mismatch is encountered, the pattern pointer is initialized to point to the pattern end, whereas the text pointer is advanced according to the greater value of the two relevant entries of the occurrence and the match table. It is important, also, to notice that Rytter proposed a correct way to calculate the match table [9]. More specifically, to each match table entry we have to add the distance of the specific character from the last pattern character.

SIMPLIFIED BOYER-MOORE ALGORITHM

This method (in short, *SBM method*) has been studied experimentally and analytically by Baeza-Yates [3]. It does not use a match table, based on the conjecture that, in practice, patterns are not periodic. A minor detail is that the occurrence table entry for the last pattern character is equal to 1 (instead of 0), to prevent the case of an infinite loop.

HORSPOOL METHOD

This method (*BMH method*) does not use a match table either [6]. In case of mismatch, the shifting size is maximized by using the occurrence table entry for the text character corresponding to the rightmost pattern character (and not for the text character where the mismatch occurred). In order to prevent an infinite loop, the occurrence table entry for the last pattern character is equal to the smallest distance from a same character in the pattern. Thus, if the last pattern character is met only once in the pattern, then this value is equal to the pattern length.

QUICK SEARCH METHOD

This method (*QS method*) was proposed recently by Sunday [11]. It is another simple (and therefore quick) method which does not use a match table but has the following two characteristics. First, comparisons start from the leftmost pattern character proceeding to the rightmost one. Second, in case of mismatch, the shifting size is equal to the occurrence table entry for the first text character after the last pattern character by the time of mismatch. For this reason, all the occurrence table values are incremented by a unit.

MAXIMAL SHIFT METHOD

This method (*MS method*), which has also been proposed by Sunday [11], uses both the occurrence and the match tables. The occurrence table is built and used in the same way as for the QS method. However, the match table is constructed in a much more complicated manner. More specifically, a temporary structure (with length equal to that of the pattern) gives, for each pattern character, the distance of an identical proceeding character in the pattern. If such a character does not exist , then the value is equal to the order of the character in the pattern. Then the pattern characters are ordered decreasingly according to this temporary table (ties are resolved in favor of the leftmost pattern character). Thus, finally we come up with a transformed pattern, for which the match table is built. The reasoning underneath this method is that, by changing the order of the pattern scan, the shift sizes are maximized when the new match table is used.

During searching, the text pointer is initialized to point to the first text character, whereas the pattern pointer points to the first position of the transformed pattern. Comparisons are performed by examining the text characters according to the order of characters of the transformed pattern. After a mismatch, the pattern pointer is initialized again, whereas the text pointer is advanced according to the maximum value of the relevant occurrence and match tables entries.

OPTIMAL MISMATCH METHOD

This is another method proposed by Sunday [11] (*OM method*), which uses both the occurrence and match tables. The occurrence table is constructed and used in the same way as for the previous two methods (QS and MS methods). The match table is built for a transformed pattern having its characters ordered ascendingly according to the frequency of appearance in the text. The reasoning behind the method is: make a mismatch as soon as possible and then make a shift as large as possible. Thus, during comparisons, priority is given to the most rare text characters.

It is evident that this method uses a temporary table during a preprocessing step, too. It is noted that, during the same period, Baeza-Yates observed independently that the pattern can be scanned in any order, and in reverse letter frequency order in particular [3].

SMITH METHOD

This is a method proposed very recently [10] (*BMS method*) and is based on the previous one. In particular, it enhances the OM method in two ways. First, it is language-independent and initially it uses a match table with arbitrary character ordering. If, during searching, a pattern character results in a mismatch, it is moved to the front of the match table so that in the next alignment, this character is tried first. Second, in case of a mismatch, we choose the greatest possible shifting size by inspecting all the occurrence table entries, which correspond to the pattern characters.

3. THE NEW METHODS

In the previous section, we have described briefly the BM algorithm and six of its variations. Their main characteristics are summarized in Table 1. As explained, all the methods use an occurrence table, although not in an identical way. Also, it is evident that it is necessary to construct an additional auxiliary table for the methods which use a transformed pattern in place of the original. We have crafted two new pattern-matching algorithms based on some of the reported techniques. The first one combines characteristics of the OM method and the BMH method; there-

TABLE 1
Characteristics of Pattern-Matching Methods Based on BM Algorithm

Method name and reference	Method code	Match table	Pattern search direction
Boyer-Moore [4]	BM	Yes	from end to start
Simplified Boyer Moore [3]	SBM	No	from end to start
Boyer-Moore Horspool [6]	BMH	No	from end to start
Quick Search [11]	QS	No	from start to end
Minimal Shift [11]	MS	Yes	character ordering
Optimal Mismatch [11]	OM	Yes	character ordering
Moyer-Moore Smith [10]	BMS	No	character ordering
Optimal Mismatch Horspool	OMH	No	character ordering
Optimal Mismatch Horspool Smith	OMHS	No	character ordering

fore, we call it, in short, the *OMH method.* The second one is based on the OHM and BMS methods; for this reason, we call it the *OMHS method*, in short. In the following subsections, we examine the new algorithms in more detail.

OHM METHOD

The code in "C" for this method can be found in the Appendix. The method resembles the BMH method since:

- it does not use a match table.
- the occurrence table entry for the last pattern character is equal to the smallest distance from a same character in the pattern, whereas if this character is met only once in the pattern, then this value is equal to the pattern length.

In addition, it has common characteristics with the OM method since:

- it uses an auxiliary table for the text character frequencies to order the pattern characters.
- it builds an auxiliary structure with the pattern characters sorted ascendingly according to the appearance frequencies and the distances from the last pattern character.
- it makes comparisons considering the end of the pattern.

The following example will demonstrate the details of the method.

EXAMPLE. Suppose we seek for the 11-character pattern "abracadabra" in the text "abracababracadabra." The occurrence table is illustrated in Table 2. Table 3 gives the frequencies of appearance for characters in an English text ([11]). The pattern characters appearing in the first row of Table 4 are ordered according to ascending frequencies as shown in Table 3. Integer numbers in the second row of Table 4 give the distance of the specific character from the last pattern character.

Searching starts by setting the text pointer to the eleventh character in the text ("a"). The first two pattern characters of the auxiliary structure (the two "b"s) match with their corresponding text characters in the

TABLE 2

Occurrence Table for the OMH Method

...	a	b	c	d	e	...	q	r	s	...
...	3	2	6	4	11	...	11	1	11	...

TABLE 3

Appearance Frequencies of Characters in English Text (%)

Character	a	b	c	d	e	f	g	h	i	j	k	l	m
Frequency	8.9	2.3	4.5	3.2	11.1	1.5	2.4	2.9	7.8	0.2	1.1	5.5	3.2
Character	n	o	p	q	r	s	t	u	v	w	x	y	z
Frequency	6.8	6.9	3.1	0.2	7.4	5.6	7.1	3.6	1.0	1.1	0.3	2.0	0.2

$(11-2=)$ ninth and the $(11-9=)$ second positions. However, there is a mismatch between the third character of the structure ("d") and the relevant $(11-4=)$ seventh text character ("b"). Thus, we have to shift the text pointer a number of positions shown by the occurrence table entry for the character "a," which is the text character corresponding to the last pattern character. Now the text pointer shows to the $(11+3=)$ fourteenth character ("d"). This time, we recognize a mismatch at the first comparison (pattern's "b" vs. text's "c"). So, we move the text pointer by four positions, as the occurrence table entry for "d" shows. At this point, we have a perfect match of the pattern and the corresponding portion of the text. Table 5 illustrates the method. Bold typed text characters of the first line denote the characters which are aligned to the last pattern character at each successive positioning. Lines 2 to 4 correspond to the three positionings of the pattern with respect to the text. In each of these lines, we show which text characters are compared, and the order of each comparison.

OMHS METHOD

This new method:

- inherits the characteristics of the previous one, and in addition,
- uses the technique proposed by Smith [10], that is, in case of mismatch we may choose the greatest shifting size by inspecting all the occurrence table entries (which correspond to the pattern characters).

TABLE 4

Auxiliary Structure with Ordered Pattern Characters. The Number is the Offset of an Occurrence of this Character, from the End of the Pattern

b	b	d	c	r	r	a	a	a	a	a
2	9	4	6	1	8	0	3	5	7	10

TABLE 5

Order of Comparisons to Text Characters in Each Pattern Positioning

Text	a	b	r	a	c	a	b	a	b	r	**a**	c	a	**d**	a	b	r	**a**
1st pattern positioning		2					3		1									
2nd pattern positioning											4							
3rd pattern positioning							15	6	10	14	8	13	7	12	5	9	11	

This technique has its own processing cost and may result in excess searching time for patterns with many different characters. After experimentation, we decided to partially adopt it by considering only two occurrence table entries: the ones for the two text characters which correspond to the last two pattern characters. The following example demonstrates this new algorithm, whereas the "C" code may be found in the Appendix.

EXAMPLE. Suppose, again, that we search for the six-character pattern "abacab" in the text "bacabadabacab." The occurrence table is illustrated in Table 6, whereas the additional auxiliary structure is depicted in Table 7.

Searching starts by initializing the text pointer to point to the sixth character in the text, whereas the pattern pointer points to the beginning of the auxiliary structure. The first character of the auxiliary structure ("b") does not match to the sixth text character ("a"). The shifting size is equal to the greater of the occurrence table entries for the two text characters ("b" and "a") corresponding to the two last pattern characters. The two entries are 4 and 1, respectively; thus, we have to advance the text points $(4 - 1 =)$ three positions, as shown by the occurrence table entry for the character "b." After one match between the text's "b" and the pattern's "b," we have a mismatch between the pattern character "b" and the text character "a." This time, the shift size is equal to the greater of the two numbers $(1 - 1 =)$ 0 and 4, which correspond to the two text characters ("a" and "b," respectively). After we advance the text pointer by

TABLE 6

Occurrence Table for the OMHS Method

...	a	b	c	d	...
...	1	4	2	6	...

TABLE 7

Auxiliary Structure with the Ordered Pattern Characters

b	b	c	a	a	a
0	3	2	1	4	5

four positions, we have a perfect match of the pattern and the corresponding portion of the text. Table 8 is similar to Table 5 and summarizes the example.

4. RESULTS AND DISCUSSION

To compare our new methods to the previous ones, we ran an experiment with a methodology similar to that of [11]. More specifically, we created an English text of approximate size 65 Kbytes with approximately 7600 distinct lexicographically ordered patterns categorized in 15 classes of length from 1 to 15 characters. We implemented all the methods using the Turbo C optimizing compiler on a 486 workstation. We greatly used the code presented in [2, 11] for the known methods. Our cost metrics were the required number of comparisons as well as the required execution time. More specifically, to understand better the merits and pitfalls of each algorithm, we counted:

- the number of *direct comparisons*,
- the number of *indirect comparisons*,
- the elapsed *searching time*, and
- the elapsed *preprocessing time*.

The number of direct comparisons (which occur by an "if ... then ... else" statement) is a classical criterion for analyzing algorithms. However, the total number of comparisons is also a crucial measure, since the number of indirect comparisons (which are performed when accessing the occurrence table) is an important overhead. It is also interesting to have a measure of the preprocessing cost, which varies significantly between

TABLE 8

Order of Comparisons to Text Characters in Each Pattern Positioning

Text	b	a	c	a	b	**a**	d	a	**b**	a	c	a	**b**
1st pattern positioning						1							
2nd pattern positioning						3			2				
3rd pattern positioning								9	8	5	6	7	4

TABLE 9
Total Number of Direct Comparisons Per Character

Length	Freq	BM	SBM	BMH	QA	MS	OM	BMS	OMH	OMHS
1	26	1.000	1.000	1.000	0.509	0.509	0.509	0.509	1.000	1.000
2	38	0.538	0.552	0.538	0.382	0.384	0.373	0.363	0.536	0.536
3	343	0.370	0.382	0.371	0.294	0.294	0.284	0.271	0.367	0.359
4	880	0.289	0.302	0.290	0.241	0.247	0.234	0.219	0.283	0.271
5	1031	0.239	0.251	0.210	0.209	0.213	0.201	0.184	0.233	0.218
6	1204	0.207	0.218	0.208	0.185	0.190	0.177	0.160	0.201	0.184
7	1192	0.183	0.193	0.183	0.166	0.170	0.159	0.141	0.177	0.159
8	963	0.164	0.173	0.165	0.152	0.154	0.144	0.127	0.158	0.140
9	780	0.150	0.159	0.151	0.141	0.142	0.134	0.115	0.145	0.126
10	533	0.140	0.148	0.141	0.132	0.133	0.125	0.107	0.134	0.115
11	336	0.130	0.138	0.132	0.125	0.124	0.118	0.099	0.126	0.106
12	166	0.122	0.130	0.123	0.117	0.117	0.111	0.093	0.117	0.098
13	90	0.117	0.124	0.118	0.112	0.112	0.107	0.088	0.112	0.093
14	39	0.111	0.117	0.113	0.106	0.105	0.100	0.083	0.106	0.087
15	13	0.104	0.109	0.106	0.101	0.099	0.094	0.077	0.100	0.082
Average	7634	0.206	0.216	0.207	0.181	0.173	0.173	0.156	0.198	0.184

simple and complicated methods. However, if the text size is large (i.e., > 10 Kb), then the preprocessing time should be ignored since the searching time is considerably larger.

Tables 9 through 12 depict our results. The values for the number of comparisons (either direct or indirect and the elapsed time (either for processing or for preprocessing) are produced in the following way: for each pattern class and metric, we have divided the measured value by the population of the class and the size of the text. Thus, all values have been normalized and denote the cost per character.

Closer inspection of the results obtained in the next four tables results in the following remarks.

- *Direct comparisons* (See Table 9). The average number of direct comparisons is calculated by taking into account the probability distribution of pattern length. The methods are ranked as follows: BMS, MS, OM, QS, OMHS, and OMH. More specifically, for small patterns, the ranking is BMS, OM, QS, and MS, whereas for larger patterns, the ranking is BMS, OMHS, and OM. This means that embedding the technique proposed by Smith in the OMH method improves the latter substantially for long patterns. Thus, in general, the Smith method is the best regardless of the pattern class. Therefore, an interesting problem is to analyze its complexity.

TABLE 10

Total Number of Comparisons Per Character

Length	Freq	BM	SBM	BMH	QS	MS	OM	BMS	OMH	OMHS
1	26	1.000	1.000	1.000	1.017	1.017	1.017	1.017	1.000	1.000
2	38	0.538	0.552	0.538	0.732	0.734	0.723	0.708	0.536	0.536
3	343	0.371	0.382	0.371	0.563	0.563	0.553	0.532	0.367	0.359
4	880	0.289	0.302	0.290	0.464	0.469	0.456	0.430	0.283	0.271
5	1031	0.239	0.251	0.240	0.400	0.403	0.392	0.362	0.233	0.218
6	1204	0.207	0.218	0.208	0.354	0.357	0.346	0.314	0.201	0.184
7	1192	0.183	0.193	0.183	0.318	0.320	0.311	0.277	0.177	0.159
8	963	0.164	0.173	0.165	0.290	0.290	0.283	0.249	0.158	0.140
9	780	0.150	0.159	0.151	0.269	0.268	0.261	0.227	0.145	0.126
10	533	0.140	0.148	0.141	0.252	0.251	0.245	0.209	0.134	0.115
11	336	0.130	0.138	0.132	0.238	0.235	0.231	0.195	0.126	0.106
12	166	0.122	0.130	0.123	0.223	0.221	0.217	0.182	0.117	0.098
13	90	0.117	0.124	0.118	0.215	0.212	0.209	0.173	0.112	0.093
14	39	0.111	0.117	0.113	0.202	0.199	0.196	0.163	0.106	0.087
15	13	0.104	0.109	0.106	0.193	0.188	0.186	0.152	0.100	0.082
Average	7634	0.206	0.216	0.207	0.346	0.348	0.339	0.307	0.198	0.184

TABLE 11

Mean Pattern-Matching Time (in MS)

Length	Freq	BM	SBM	BMH	QS	MS	OM	BMS	OMH	OMHS
1	26	0.115	0.100	0.099	0.082	0.151	0.145	0.112	0.154	0.197
2	38	0.059	0.053	0.052	0.059	0.107	0.102	0.078	0.082	0.104
3	343	0.041	0.037	0.036	0.045	0.082	0.078	0.059	0.056	0.071
4	880	0.031	0.029	0.028	0.037	0.068	0.065	0.048	0.043	0.053
5	1031	0.026	0.024	0.023	0.032	0.059	0.056	0.040	0.036	0.043
6	1204	0.022	0.021	0.020	0.029	0.052	0.049	0.035	0.031	0.036
7	1192	0.020	0.018	0.017	0.026	0.047	0.044	0.031	0.027	0.032
8	963	0.018	0.017	0.016	0.023	0.042	0.040	0.028	0.024	0.028
9	780	0.016	0.015	0.014	0.022	0.039	0.037	0.025	0.022	0.025
10	533	0.015	0.014	0.013	0.020	0.036	0.035	0.023	0.021	0.023
11	336	0.014	0.013	0.013	0.019	0.034	0.033	0.022	0.019	0.021
12	166	0.013	0.012	0.012	0.018	0.032	0.031	0.020	0.018	0.020
13	90	0.012	0.012	0.011	0.017	0.031	0.030	0.019	0.017	0.019
14	39	0.012	0.011	0.011	0.016	0.029	0.028	0.018	0.017	0.017
15	13	0.011	0.011	0.010	0.016	0.028	0.027	0.018	0.015	0.016
Average	7634	0.022	0.021	0.020	0.028	0.051	0.048	0.034	0.031	0.036

TABLE 12

Mean Preprocessing Time (in MS)

Length	Freq	BM	SBM	BMH	QS	MS	OM	BMS	OMH	OMHS
1	26	0.047	0.012	0.041	0.029	0.049	0.049	0.036	0.044	0.011
2	38	0.051	0.012	0.041	0.031	0.064	0.071	0.052	0.050	0.019
3	343	0.054	0.013	0.042	0.031	0.086	0.103	0.077	0.061	0.029
4	880	0.058	0.013	0.042	0.033	0.127	0.187	0.151	0.075	0.043
5	1031	0.062	0.014	0.043	0.034	0.143	0.225	0.183	0.094	0.061
6	1204	0.066	0.014	0.043	0.035	0.175	0.289	0.238	0.116	0.083
7	1192	0.069	0.015	0.044	0.037	0.199	0.350	0.291	0.142	0.108
8	963	0.073	0.015	0.044	0.038	0.240	0.416	0.350	0.171	0.137
9	780	0.076	0.016	0.045	0.039	0.276	0.480	0.407	0.205	0.170
10	533	0.080	0.016	0.045	0.041	0.309	0.548	0.466	0.242	0.207
11	336	0.084	0.017	0.046	0.042	0.345	0.619	0.531	0.283	0.247
12	166	0.086	0.018	0.047	0.043	0.380	0.695	0.600	0.328	0.292
13	90	0.091	0.018	0.047	0.044	0.418	0.756	0.654	0.375	0.338
14	39	0.094	0.019	0.047	0.046	0.463	0.814	0.707	0.430	0.392
15	13	0.098	0.019	0.049	0.046	0.501	0.930	0.809	0.489	0.450
Average	7634	0.069	0.015	0.044	0.037	0.209	0.354	0.296	0.151	0.118

• *Indirect comparisons* (See Table 10). The methods QS, MS, OM, and BMS perform indirect comparisons, whereas the remaining methods BM, SBM, BMH, OMH, and OMHS) do not. In other words, the relative weight of the advantage of the Smith method is decreased. Thus, on the average, the ranking according to the total number of comparisons is OMHS, OMH, BM, BMH, and SBM. Let us notice that this ranking does not change with the pattern length. It is also interesting to note that the new methods OMH and OMHS outperform the BMH, OM, and BMS methods, although they are chemical unions of the latter ones.

• *Searching time* (See Table 11). Again, the average searching time is calculated by taking into account the probability distribution of pattern length. The ranking is BMH, SBM, BM, QS, OMH, BMS, and OMHS. However, for long patterns, it seems that (a) the performance of the QS and OMH algorithms is very similar, and (b) the new OMHS method outperforms the S method. We also remark that the embedding of the Horspool technique in the OM method is successful performance-wise. This remark does not hold for the embedding of the Smith technique in the OMH method.

• *Preprocessing time* (See Table 12). It is evident that it increases with increasing pattern length. After averaging, we conclude that the ranking sequence is SBM, QS, BMH, BM, OMHS, and OMH. As expected, this ranking is very similar to the ranking according to the searching time metric. We also remark that the cost increase for the patterns of length 15 in comparison to the cost for patterns of length 1 is very small (ratio less than 1:2) for the BMH, QS, and SBM methods. This observation does not hold for the other methods. For example, for the BMS and OMHS methods, this ratio is greater than 1:20.

Concluding, in this paper we have presented the results of an extensive experiment of the most well-known pattern-matching algorithms based on the Boyer-Moore method. We have described two new variations (the OMH and OMHS methods) by building blocks of previous variations. Summarizing the previous comments with regard to the new methods, we have the following.

- From the theoretical point of view, the OMHS method is always the best method, since the total number of comparisons is the most important metric in such a case. If we are interested in the number of direct comparisons, the OMHS method is second best after the BMS method (for length > 6).
- From the practical point of view, thanks to their simplicity, the BMH and SBM methods seem to be very stable methods, outperforming the other variants in terms of searching time.

APPENDIX

The names of the variables used in the following fragments of code are summarized in Table 13.

OMH METHOD

```
typedef struct pattern_scan_element
typedef {
        int loc; char c;
        }
        aux;
```

TABLE 13

Names of Variables Used and Definitions

Variable	Definition	Variable	Definition
text	text pointer	ASIZE	alphabet size
tlen	text length	occ	occurrence table
pattern	pattern pointer	freq	frequency table
plen	pattern length	aux	auxiliary structure
MAXPAT	maximum pattern length	auxptr	auxiliary table

```
static aux auxptr[MAXPAT];
void prep(unsigned char *pattern, unsigned char plen)
{
  unsigned char h; register int i;
  register unsigned char *c, *ce; register aux *p, *p1, *ps;
  for (i=0; i<ASIZE; i++) occ[i]=plen;
  for (c=pattern, ce=pattern+plen-1; c<ce; c++) occ[*c]=ce-c;
  for (i=plen-1, p=auxptr; i>=0; i--, pattern++, p++)
  {
    p->loc=i; p->c=*pattern;
  }
  p->c=0; p->loc=0;
  for (p=auxptr+plen-1, ps=auxptr+1; p>=ps; p--)
  {
    for (p1=p-1; p1>=auxptr; p1--)
    {
        if (freq[p1->c-97]<freq[p->c-97]) continue;
        h=p->c; i=p->loc; p->c=p1->c;
        p->loc=p1->loc; p1->c=h; p1->loc=i;
    }
  }
}

long OMHsearch(unsigned char *text, unsigned int tlen)
{
  register aux *p; register unsigned char *tx;
  register unsigned int found;
  tx=text+plen-1; found=0;
  while (tx<=text+tlen)
  {
    for (p=auxptr; p->c; p++) if ( p->c != *(tx-p->loc) ) goto
mismatch;
    found++;
    mismatch: tx+=occ[*tx];
  }
  return found;
}
```

OMHS METHOD

```
long OMHSsearch(unsigned char *text, unsigned int tlen)
{
  register aux *p; register unsigned char *tx;
  register unsigned int found;
  tx=text+plen-1; found=0;
  while (tx<=text+tlen)
  {
    for (p=auxptr; p->c; p++) if ( p->c != *(tx-p->loc) ) goto
mismatch;
```

```
      found++;
      mismatch: tx+=(occ[*tx]>occ[*(tx-1)]-1 ? occ[*(tx-1)]-1);
    }
    return found;
}
```

This research was performed while Y. Manolopoulos was on sabbatical leave at the University of Maryland at College Park and the Institute for Systems Research (ISR). The research of C. Faloutsos was partially funded by the Institute for Systems Research (ISR), by the National Science Foundation under Grants IRI-9205273 and IRI-8958546 (PYI), with matching funds from EMPRESS Software Inc. and Thinking Machines Inc. Thanks are due to Mr. Apostolos Papadopoulos and Mr. Michael Timoleon for their valuable assistance in the course of this effort.

REFERENCES

1. A. V. Aho, Algorithms for finding patterns in strings, in J. VanLeeuwen (ed.), *Handbook of Theoretical Computer Science, Volume A: Algorithms and Complexity*, Elsevier, Amsterdam, The Netherlands, 1990, pp. 255–300.
2. R. A. Baeza-Yates, Algorithms for string searching—A survey, *ACM SIGIR Forum* 23(3–4):34–58 (1989).
3. R. A. Baeza-Yates, Improved string searching, *Software—Practice and Experience* 19:257–271 (1989).
4. R. S. Boyer and J. S. Moore, A fast string searching algorithm, *Comm. ACM* 20(10):762–772 (1977).
5. C. Faloutsos, Access methods for text, *ACM Comput. Sur.* 17(1):49–74 (1985).
6. R. N. Horspool, Practical fast searching in strings, *Software—Practice and Experience* 10:501–506 (1980).
7. A. Hume and D. Sunday, Fast string searching, *Software—Practice and Experience* 21(11):1221–1248 (1991).
8. D. E. Knuth, J. H. Morris, and V. R. Pratt, Fast pattern matching in strings, *SIAM J. Comput.* 6(2):323–349 (1977).
9. W. Rytter, A correct preprocessing algorithm for the Knuth-Morris-Pratt algorithm, *SIAM J. Comput.* 9:509–512 (1980).
10. P. D. Smith, Experiments with a very fast substring search, *Software—Practice and Experience* 21(10):1065–1074 (1991).
11. D. Sunday, A very fast substring search algorithm, *Commun. ACM* 33(4):132–142 (1990); *Commun. ACM* 35(4):132–137 (1992).

Received 1 March 1995; revised 11 August 1995