Hindawi
Mobile Information Systems
Volume 2022, Article ID 6521905, 11 pages
https://doi.org/10.1155/2022/6521905

Research Article

An Information Retrieval Algorithm for Accounting Internal Audit Using Multi-Pattern Similarity Matching

Chunguang Ma,[1] **Hongjun Bei**,[1] **Guihua Chen**,[2] **and Jianhui Gao**[3]

[1]*Ningbo University of Finance & Economics, Ningbo 315010, China*
[2]*Ningbo Daocheng Construction Services Co. Ltd, Ningbo 315010, China*
[3]*Ningbo Raw Water Co. Ltd, Ningbo 315010, China*

Correspondence should be addressed to Chunguang Ma; machunguang@nbufe.edu.cn

Received 31 March 2022; Revised 11 May 2022; Accepted 28 May 2022; Published 16 June 2022

Academic Editor: Muhammad Babar

The multi-mode matching has noteworthy transformations equated with the classical multi-mode matching algorithms. It is frequently used for the policy part of the TCP connection to connect the English characters. In this article, we analyzed the features of multi-mode similarity for audit information retrieval in a cluttered environment. The proposed model analyzed the performance theorem of a multi-mode matching algorithm for audit information retrieval. It also analyzed the shortcomings of existing multi-mode similarity systems and proposed a multi-mode algorithm based on the trail hash trie matching machine suitable for mixed Chinese and English environments. The algorithm converts the set of pattern strings into multiple finite automata and then builds a state driver using the set of pattern strings. The state driver is driven by the characters of the string to be matched in turn, and each finite automaton is driven by the state driver to achieve similar multimodal matching with mixed English and Chinese characters by allowing the insertion errors. The algorithm does not need to match every character and can make full use of the information of this unsuccessful match during the matching process and skip as many characters as possible by combining the improved text window mechanism. It can control the upper limit of allowed errors for each pattern string. The matching speed is independent of the number k of allowed insertion errors. The algorithm has comprehensive application projections in the fields of information auditing, database, and information retrieval, respectively.

1. Introduction

The Internet has now been one of the most essential global information channels. In contrast, the application of information processing technologies such as network information retrieval and network information content auditing has become more and more widespread, and multi-mode matching algorithms are the core of these technologies, and most of the problems related to network information processing are eventually converted into multi-mode matching problems [1, 2]. The multi-mode matching for web information text has significant differences compared with the traditional multi-mode matching. It is frequent for the policy part of the TCP connection to consist of English characters, whereas the content-related part of the packet will consist of other types of characters for a variety of reasons, along with Internet protocol encoding and language groups in different regions [3, 4]. Hence, the text to be processed is usually composed of a mixture of characters with different encoding rules. In the case of Chinese, this situation is more prominent because of the difference between simplified and traditional Chinese characters, and the text will comprise 2 or more types of characters. The multi-mode matching is oriented to a single-character environment, and when applied to the abovementioned mixed environment, there are certain challenges including missed matching and mismatching [5, 6]. Therefore, it is of great theoretical and application value to study efficient and practical multi-mode matching algorithms for a mixed environment.

In this study, we introduce THT (threaded hash trie), a multi-pattern technique based on a threaded whole hash trie matching machine. The THT algorithm is ideal for multi-pattern matching systems in mixed foreign contexts, with low-time complexity and tolerable space complexity. The characteristics of multi-mode matching for audit information retrieval are analyzed. The proposed solution also analyzed the shortcomings of existing multi-mode matching algorithms using the THT algorithm. The algorithm converts the set of pattern strings into multiple finite automata. The algorithm does not need to match every character and can make full use of the information. The algorithm has broad application prospects in the fields of information auditing, database, and information retrieval.

The rest of this article is organized as follows: In Section 2, the existing approaches used for multi-mode matching are presented. In Section 3, various theorems for multi-mode matching in a mixed environment are presented. Section 4provides a solid foundation for our threaded hash trie algorithm. In Section 5, the experimental results and analysis of the THT algorithm are presented. Finally, the study is concluded in Section 6.

2. Existing Approaches for Multi-Mode Matching

A specific character is encoded as ASCII (American Standard Code for Information Interchange) in the English language that occupies only one byte. On the other hand, Chinese language character is encoded for double, simplified characters as GB (gigabyte), with the greatest bit of each byte set to 1. The traditional characters are stored as BIG5 with the highest bit set to 0. The following characteristics are allowed for a byte in mixed text. The byte with both the highest bit of 0 is an English character; the byte with the highest bit of 1 is a simplified or conventional Chinese [7]. This difference in encoding length and rules makes multi-mode matching in a mixed environment extremely complicated. In a single-character environment, the encoding length of characters is the same, and the matching algorithm only needs to match according to the fixed length and can speed up the matching by jumping according to the rules.

However, due to the randomness of network information, the probabilities of different coded characters in text strings are random in mixed environments. The problem of mismatching will arise if the attributes of the bytes are incorrectly determined during the matching process, resulting in a series of matching errors by comparing the low byte of the Chinese letter in the text string with the high byte of the Chinese character in the pattern string, or by mistakenly matching the low byte of a traditional Chinese letter with an English letter. For example, for the character string "Expense," (3C|62|3E|CBD1|CBF7|B2FA|C6B7|3C|2 F| 62| 3E), assuming "product (B2FA| C6B7)" is the keyword. If there is an error in processing the character boundary, the match starts from the lower byte D1 of "search" and the combination of bytes becomes "3C|62|3E|CB|D1CB|F7B2|FAC6|B73C|2F|62|3E," which will clearly generate a series of mismatches and lead to a missed match; if the mismatched codes exactly form a keyword, it will lead to a false match. The mismatch will not be corrected until a non-Chinese character appears if the Chinese character is followed by a Chinese pattern string after a byte mismatch occurs. The mismatch will be caused if the low byte of a Chinese letter and the high byte of the adjacent Chinese character happens to form a pattern string. To prevent mismatching, the common method is to filter out the non-Chinese characters in the text string, convert it into a pure Chinese text string, and then process it accordingly. Obviously, this requires two scans of the text string, which wastes more system time and has a lower matching efficiency. This has a great impact on real-time processing such as web information content audit and content search. A few Chinese scholars have conducted some research on pattern matching in Chinese and mixed Chinese/English environments in response to the above problems.

These studies have made very useful explorations on pattern matching in mixed Chinese/English environments, but they all have certain shortcomings. For countries that mainly use single-byte encoding (e.g., US and UK), there is less relevant literature because the problems discussed above do not exist. When applied directly to Chinese character matching, the classical DFSA algorithm available in the literature [1] has excellent efficiency whenever compared to the English character habitat but increases the storage space when applied to an English character environment. The state transition function is typically stored in a complete hash table of high access efficiency to achieve high matching speed. The same space taken up is sum_of (states) 256 that is 256 times more than the space occupied because a lot of states rises if the same method is being used for Chinese characters. The number of products that are required for the whole hash table grows rapidly, making the method impractical for direct application in practice. In the literature [6], the classical multi-pattern matching algorithm DFSA is applied to Chinese character matching, and the problem of storage space expansion is proposed to construct a combinatorial state automaton by decomposing the inner code of Chinese characters and using the idea of QS algorithm for acceleration. The algorithm solves the space expansion problem when constructing a complete hash table for Chinese characters, but it is only applicable to the pure Chinese character environment, which will lead to byte misalignment problem in the mixed Chinese and English environment. The "tagging" method is used to prevent the misalignment problem in matching, that is, the attributes of each bit in the pattern string and the text to be matched are tagged [7, 8]. The type of each bit can be divided into 3 categories that include (Type 0) ASCII for English, (Type 1) high byte for Chinese characters, and (Type 2) low byte for Chinese characters.

It is important that two bytes not only have the similar value but had the same mark to be regarded a successful match when 2 bytes are compared. In mixed Chinese and English contexts, this method can be used to solve the problem of byte mismatches; nevertheless, it is less productive, so it needs or before the text string to be aligned to mark each byte, and it does not take account the lawsuit of

mixed ACSII, GB, and BIG5 compression algorithms. A proposal is designed that maps all Chinese letters into a set of size 65 536 by hashing both high byte Hbyte and low byte Lbyte of a internal code in string Chinese letter, such as $256 \times \text{Hbyte} + \text{Lbyte}$ [9]. A two-level hash table-based DFSA case is proposed. The algorithm uses two bytes of Chinese characters as a minimum matching unit, which can avoid the byte mismatch problem in the mixed Chinese and English environment. It is also applicable to the case of three mixed encodings, but the hash mapping operation must be performed for each Chinese character in the text string to be matched, which will undoubtedly affect the matching speed of the algorithm. As has been seen in fact, this issue has a substantial impact on the overall efficiency of the method. In addition, some other papers have also studied multi-mode matching algorithms, but they do not consider the impact of mixed Chinese and English environments on the accuracy and matching efficiency of the matching algorithm [e.g., 4, 10]. The study of an efficient and practical multi-mode matching algorithm suitable for the mixed environment of multiple coding characters is extremely important for the development of accounting internal audit information retrieval technology and has a large theoretical and application value [11].

3. Theorems Related to Multi-Mode Matching in Mixed Environment

The multi-mode matching in a mixed environment has its own characteristics compared with a single-character environment as from the previous analysis. Since the number of bytes occupied by English characters and Chinese characters is different, two bytes of data need to be compared to match two English characters, while four bytes of data need to be compared to match two Chinese characters. To facilitate the discussion later, the definitions of the number of matches and the number of comparisons are given.

3.1. Definition: Byte Length vs. Character Length. Suppose T is a random text string containing Chinese and English characters, where the number of Chinese characters is m and the number of English characters is n. The character length of T is defined as the sum of the number of Chinese characters and English characters contained in T, that is, $m + n$. The byte length is the number of bytes occupied by T in memory, that is, $2m + n$. For pattern matching in an English character environment, the number of matches and comparisons are equal, and for an English text string, the length of characters and bytes are also equal. However, for multi-byte inner code characters, there is a multiplicative relationship between the inner code lengths.

Theorem 1. *If the set of pattern strings contains a pattern string of Chinese characters, the number of matches required to correctly complete the matching of any random text string containing Chinese and English characters in any case is not less than the character length of the text string.*

Proof. Let T [1, 2, ..., M] be a random text string containing Chinese and English characters to be matched, where M is its byte length, N is its character length, $T[i]$ $(1 \le i \le M)$ is a byte of the text string, and Σ is the set of pattern strings [4]. Whether to preprocess the text T to be matched or not, the matching algorithm can be divided into two categories: one preprocesses T, and the other does not process it and matches it directly. In contrast, the proof of the theorem is also divided into two parts. □

3.1.1. Preprocessing of T. Whether the bytes in T are tagged or the English or Chinese characters in T are removed and converted into a single-character text string, even if only the high byte of the Chinese character is discriminated (to determine whether the highest bit is 1), the pre-scan requires at least N comparisons to determine the correct attribute of each byte $T[i]$ $(1 \le i \le M)$ in T. Therefore, for this type of algorithm, the theorem holds. Since the matching must be done after preprocessing, the number of correct matches that can be done by such algorithms is obviously larger than the character length N of T. Therefore, for such algorithms, the theorem holds.

3.1.2. No Preprocessing for T. Suppose there exists a matching algorithm A that does not preprocess, and the number of character matches Cmp required to correctly complete the matching for (Σ, T) in any case is less than the character length N of T, that is, $\text{Cmp} < N$. Since $\text{Cmp} < N$, it is obvious that in the matching process, algorithm A does not match some characters in T, that is, algorithm A makes at least 1 jump in the matching process. Let the characters skipped by algorithm A be $T[i, j](1 \le i \le j \le M)$ and assume that there is no mismatch before the jump. According to the assumptions, $j-i \ge 1$, $T[i]$, $T[i+1]$,..., $T[j]$ are random characters, and the next matching starts from $T[j+1]$ after the jump, then there are two cases, which are as follows:

(i) $j-i$ is an even number. Since $T[i]$, $T[i+1]$,...,$T[j-1]$ are random characters, then the number of English characters enum is also random, which can be either odd or even. When enum is odd, then $T[j]$ must be an English letter or the high byte of a Chinese letter. However, when $T[j]$ is the high byte of a Chinese letter and $T[j+1]$ is the low byte of the Chinese character, matching from $T[j+1]$ will obviously bring a series of matching mismatches, so in this case, the algorithm cannot guarantee any byte mismatch, that is, it cannot guarantee a correct match in all cases.

(ii) $j-i$ is an odd number. In this case, in order to ensure that no byte mismatch occurs when matching at $T[j+1]$, there must be an odd number of English characters in $T[i]$, $T[i+1]$, ..., $T[j-1]$, so that $T[j+1]$ can be the high byte of English characters or Chinese characters, and matching from $T[j+1]$ will not. This is also contradictory to the premise that T is a

random text string. Therefore, in this case, correct matching is also not guaranteed.

Combining the above two cases, the assumption does not hold, that is, there is no such matching algorithm such that for (Σ, T), the number of character matches required to complete the matching correctly in any case is less than the character length of T. Therefore, the theorem holds for the direct matching class of algorithms. The above analysis shows that the theorem holds.

Theorem 2. *In the mixed English-Chinese environment, the multi-mode matching algorithm based on jump matching has mismatched without pre-scanning the text string to be matched.*

Proof. In the mixed English-Chinese environment, the text string to be matched can be a random text string containing both English and Chinese characters. The algorithm based on jump matching skips some characters in the text string during the matching process, which is a sublinear matching algorithm, and the number of matches in the matching process will be smaller than the length of characters in the text. According to Theorem 1, in a mixed English-Chinese environment, the number of matches is less than the length of the text string to be matched, and then there is a byte mismatch problem, resulting in a missed match or a false match. □

4. Threaded Hash trie (THT) Algorithm

This section gives a concrete implementation of the THT algorithm. trie structure is a multilayer tree index structure with varying depth, which uses the width-first search strategy and finds the leaf nodes on the same level one by one from left to right, and then moves to the next level after finding the matching item. In the matching process, the search pathway is a single search, which has similar search efficiency as DFSA, but the conventional trie matching structure must traverse the leaf nodes, which reduces the matching speed. In this study, we extend the conventional trie structure by setting all trie leaf nodes into a fully hashed table of 256 size and create a fully hashed engine using the inner code of the design stream of letters as the key value. Chinese characters are represented by two levels of adjacent leaf nodes in the fully hashed engine and the English letters are represented by one level of leaf nodes. The end of the pattern string is represented by a special character. For example, the inner code of the Chinese character "Hua" [12] is 0xBBAA, and the corresponding fully hashed engine node is as depicted in Figure 1.

It is probable to realize a full hash lookup without any additional operations on the high and low bytes of the Chinese character since the maximum value of a single byte is 255. It has a very high lookup productivity as the high and low bytes of the Chinese character are constructed separately and there is no space expansion problem. The fully hashed trie matching machine is constructed in the smallest unit of bytes and can construct the Chinese pattern string and the English pattern string in the same trie matching machine

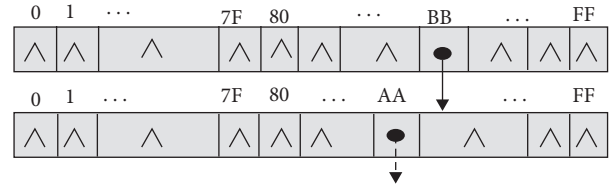

Figure 1: Tree node of Hash trie.

that has good compatibility between Chinese and English. For example, a fully hashed trie matching machine based on the set of pattern strings {Company, Shareholder, Employee} is shown in Figure 2. Figure 2 shows that the inner codes of "Company", "Shareholder," and "Employee" are 0xD6D0, 0xAABB, and 0xB9FA respectively.

The number of characters in the accounting audit information table is large, unlike English characters. The probability of failure of the first character matching is higher in the matching process. Therefore, the first character complete hash table can be used to speed up the matching, that is, only the first character is matched successfully, and then it enters the full hash trie matching machine for matching. The first character complete hash table uses a 256×256 2-dimensional array (e.g., head_index[Hbyte] [Lbyte]) to achieve a faster matching speed by directly indexing the high and low bytes of Chinese characters in the matching process.

The data structure and the pseudocode of the construction algorithm required to construct the full hash trie matching machine are shown in Algorithm 1. From Theorem 2, it is known that in the mixed Chinese and English environment, the algorithm with jump matching may produce false matches or missed matches. The algorithm governs the kind of characters prior to matching, and the match pointer advances only 1 character from the initial position of the previous match after a failed or successful match. The brute force matching algorithm is not very efficient since the matching pointer is backtracked during matching, but it can dodge missed matches or false matches.

The pseudocode of the THT matching process is shown in Algorithm 2. It should be noted that since findex [kw.num].n in Algorithm 2 is the smallest unit of measure in bytes, while n in is the smallest unit in characters, the initialization of findex in the specific program should be converted and the Chinese and English pattern strings should be distinguished.

5. Performance Analysis of the Algorithm

The matching times of the THT algorithm are close to the theoretical lower limit of the minimum number of matches required for a correct match in the mixed Chinese and English environment according to Theorem 1. Therefore, the algorithm has a good matching efficiency. In the worst case, the number of matches required to complete the matching of T is Twlen + MAXLEN and the number of comparisons is Tblen + MAXLEN. Hence, the time complexity of the algorithm is O (Tblen + MAXLEN). Moreover, the probability of matching failure is very high after matching the first high

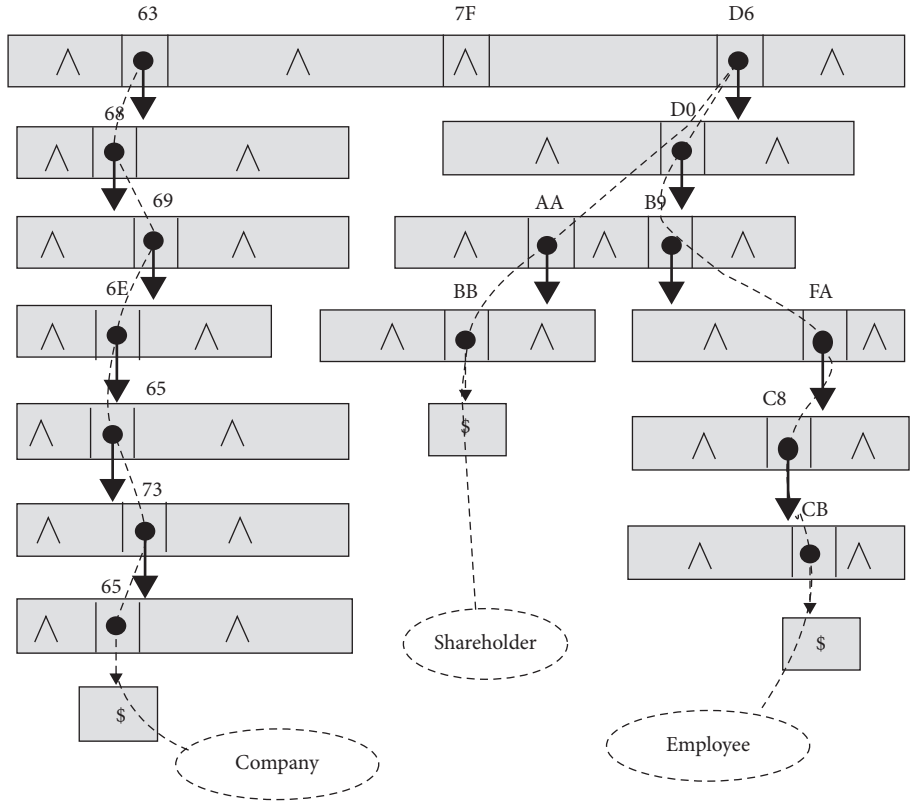

Figure 2: A hash trie matching machine.

byte of the first character since Chinese characters are a large character set. Hence, in practice, the number of comparisons of the THT algorithm is generally much smaller than the byte length of the text to be matched.

In addition, the matching speed of the THT algorithm is related to the following factors:

(1) The number of identical characters between pattern strings, that is, the number of times the characters in the set Ψ appear in the pattern string. The lower the number of identical characters, the lower the density of clues in the constructed trie matching machine, the lower the number of transfers required in matching, and the higher the matching efficiency.

(2) The probability of occurrence of a character in the set Ψ in the text string T. Obviously, the lower the probability of occurrence, the higher the probability of match failure after the first character hash matching, and the lower the matching time will be spent if the matching pointer is directly shifted back.

(3) The proportion of Chinese and English pattern strings in Σ and the proportion of Chinese and English in T. There are obvious differences between Chinese and English languages, for example Chinese language is a large character set language with a large alphabet and short words, while the English language has a small alphabet and long words. These differences make that in most cases, Chinese characters in T do not belong to Ψ, and even if they belong to Ψ, the probability of needing to match transfer after successful or unsuccessful matching is small; for English, the opposite is true. Therefore, when there are fewer English pattern strings in the set of pattern strings and fewer English characters in the text string to be matched, the algorithm matches faster.

5.1. Experimental Results and Analysis. The experimental analysis is performed, and the results are shown in this section. The experimental results are carried out in the context of text. The results are also highlighted about the space of the algorithms utilized in this research. The comparative analysis of the proposed model is provided with other algorithms.

5.1.1. Experimental Text and Results. Five groups of Chinese keywords were cut out from Text 1, and the numbers of keywords were 500, 1,000, 1,500, 2,000, and 2,500, each with an average length of 3.7 Chinese characters, whereas five groups of English keywords were cut out, and the numbers were 10, 20, 30, 40, and 50; therefore, no single-word keywords were set in the experimental process.

The main characteristics of audit data are reflected in the technique of sales, return, and the purchase stock of emotions as shown in Figure 3. The spaces used by the three algorithms are shown in Table 1, where DFD and SHENL represent the algorithms in the literature [6, 9], respectively. The matching time is significantly longer than the other algorithms since the algorithm in [8] needs pre-scanning.

```
typedef struct Head_Hash_Table{
      struch Trie_Node *head;
}Head_Index[CI\ODE_LEN][CODE_LEN];
typedf struct Trie_Node{
      struch Trie_Node *next[CODE_LEN];
}Trie_Node;
for(i = 0; i < key_sum; i++){
      if(kw[i][0]>128){
            len = strlen(kw[i]);
            if(!head_index[kw[i]][0][kw[i][1]].head){
                  p_1 = malloc(sizeof(struct Trie_Node));
                  head_index[kw[i]][0][kw[i][1]].head = p_1; }
            else
                  p_1 = head_index[kw[i]][0][kw[i][1]].head;
            for(k = 2; k < len; k++){
                  if(k< len-1){
                  if(!p_1⟶next[kw[i][k]]){
                        p_2 = mallow(sizeof(struct Trie_Node));
                        p_1⟶next[kw[i][k]] = p_1 = p_2; }
                  else p_1 = p_1⟶next[kw[i][k]]; }}
            else{
}
for(i = o; i < strlen(Text)){
      if(T[i]>128){
         fwd_num = 2;
         p = head_index[T[i]][T[i+1]].head;
         if(p&&p⟶next[T[I+2]]){
            p-p⟶next[T[i+2]]⟶next[T[i+3]];
            m = 4;
            if(End_FLAG = = p)PRINTkw;
            else
                  while (NIULL! = p){
                        if(End = = P){PRINTkw; break; }
                        if(p⟶next[T[i + m]]){p = p⟶next[T[i + m]]⟶next[T[i + m + 1]]; m + 2; }
                        else break;
            else{
                  fwd_num1; }
            i+ = fwd_num;
}
```

Algorithm 1: Complete hash trie matching algorithm.

Therefore, no comparison experiment with the algorithm in [8] is conducted in this study. To compare the theoretical value of this algorithm with the actual space used, we assume that the maximum keyword length is 10 Chinese characters and 20 English characters, and the theoretical space used by the THT algorithm is also given in Table 2.

Firstly, we use Text 2 and Text 4 for comparison. Since SHENL is only applicable to the pure Chinese environment, using Text 2 and Text 4 can ensure that each algorithm can perform correct matching, and it is meaningful to compare the efficiency in this case. Text 2 and Text 4 are pure Chinese text, so there is no need to distinguish the type of characters in the matching process. In this way, the character processing of the four algorithms is the same, and the matching time reflects the efficiency of the algorithms. During the experiments, all four algorithms were able to complete the search correctly, and the five sets of experiments for Text 2 had 11,067, 18,525, 22,987, 30,210, and 50,181 matches; the five sets of experiments for Text 4 had 379,016, 451,942, 609,310, 941,736, and 1,636,960 matches; the time used by each algorithm is shown in Table 1.

The performance of the four algorithms, DFD, SHENL, HT, and THT, was evaluated using Text 1 and Text 3 in a mixed English and Chinese environment, and the performance comparison is shown in Table 3. In the mixed English-Chinese environment, SHENL could not perform correct matching, and the 5 sets of experimental matches for Text 1 were 5,683, 9,394, 11,805, 15,426, and 31,092; the 5 sets of experimental matches for Text 3 were 189,328, 224,678, 314,580, 489,099, and 1,053,692. Table 4 shows the number of comparisons between algorithm HT and THT algorithm in real operation.

5.1.2. Analysis of Experimental Results. We can see that the space required by this algorithm is less than that required by

```
for(i = 0; i < strlen(T)){
  if(T[i]>128){ //Chinesh word
    p_1 = head_index[T[i]][T[i+1]].head;
    if(p_1and&p_2 = p_1⟶next[T[i+2]]){
      if(!p_1 = p_2⟶next[T[i+3]]) i+ = 2;
      else{
        m = 4;
        while(p_1){
          if(END_)FLAG = = p_1) {
            PRINTkw;
            if(sindex[kw.num].index = = NULL){i+ = m; break; }
            Else{
              p_1 = sindex[kw.num].index; i = i + m-sindex[kw.num].n;
              m = sindex[kw.num].n; }}
        if(p_2 = p_1⟶next[text[i + m]]){
          p_1 = p_2⟶next[text[i + m+1]];
          if(p_1 = = NULL){
            if(findex[kw.num].index = = NULL){i = i + m; break; }
            else{p_1 = findex[kw.num].index; i = i + m–findex[kw.num].num = findex[kw.num].num; }}
            m+ = 2}
          else{
              if(findex[kw.num].index = = NULL){i = i + m; break; }
              else{p_1 = findex[kw.num].index; i = i + m–findex[kw.num].n; m = findex[kw.num],n}
      }}}}
      else i+ = 2; }
  else{ //English char}
  }
```

Algorithm 2: THT matching algorithm pseudocode.

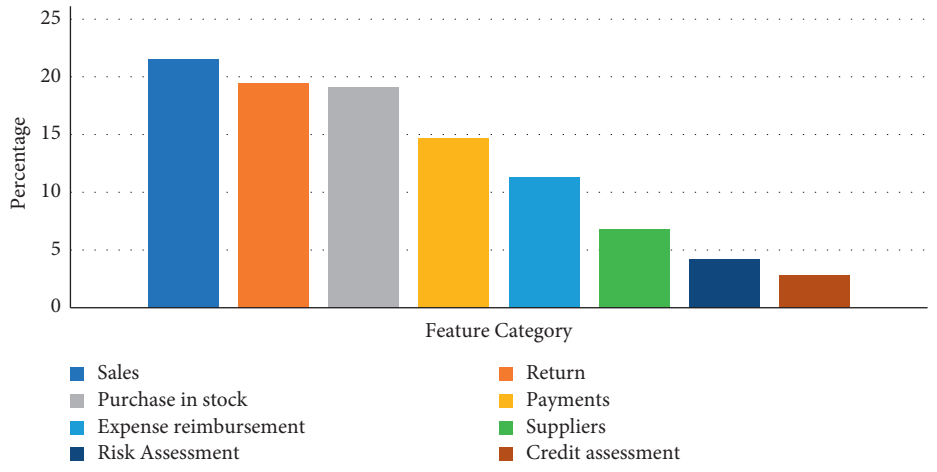

Figure 3: Types and percentages of audit data characteristics.

Table 1: Time used by each algorithm.

Number of keywords	Text 2 (s)				Text 4 (s)			
	DFD	SHENL	HT	THT	DFD	SHENL	HT	THT
500	0.187	0.084	0.061	0.077	5.187	2.824	2.261	2.572
1000	0.237	0.120	0.082	0.099	5.733	3.921	2.862	3.394
1500	0.265	0.145	0.097	0.101	6.255	4.645	4.177	3.881
2000	0.327	0.212	0.118	0.116	7.613	5.613	4.658	4.415
2500	0.363	0.321	0.131	0.121	8.143	6.921	5.351	4.931

Table 2: Space of the algorithms.

Number of keywords		Space used (Mb)			
Audit	Search	DFD	SHENL	HT	THT theoretical value
501	12	1.36	0.88	0.74	2.41
1001	22	3.32	1.53	1.27	4.65
1501	32	5.71	2.16	1.77	6.81
2001	42	8.36	2.77	2.25	9.14
2501	52	10.76	3.36	2.71	11.38

Table 3: Algorithms' matching time on mixed texts.

Number of keywords	Text 2 (s)			Text 4 (s)		
	D.F.D	H.T	T.H.T	D.F.D	H.T	T.H.T
501	0.193	0.086	0.117	6.187	2.462	3.532
1001	0.256	0.121	0.156	7.733	2.878	3.491
1501	0.280	0.202	0.176	9.255	4.326	3.782
2001	0.339	0.227	0.248	10.613	4.811	4.242
2501	0.401	0.271	0.292	10.143	5.429	4.996

Table 4: Comparing times of HT and THT.

Keywords	Text 1		Text 3	
	H.T	T.H.T	H.T	T.H.T
500	2865058	2948630	143621794	145737870
1000	3166592	3212740	150451290	156107295
1500	3324754	3355030	154931714	153743596
2000	3597439	3482754	160420458	158121882
2500	3947920	3558562	176533172	163136583

DFD and SHENL algorithms from Table 1 and Figure 4. The space required by the DFD algorithm increases and becomes superlinear as the number of pattern strings increases. Along with space usage, we discuss a number of other performance metrics in this section.

(1) Space usage. The space required by this algorithm and SHENL algorithm increases primarily with the number of pattern strings and becomes sublinear. This is due to the fact that the DFD algorithm makes use of a complete hash table of states, and the number of states in the state machine grows more rapidly as the number of pattern strings grows. The actual space used by the THT method is far less than the space required by the theoretical calculation, which is consistent with the conclusion of the performance analysis for this algorithm.

(2) Matching time. In terms of matching time, a comparison between THT and other algorithms is presented in Figure 5. This comparison takes into account the time required for matching the keywords.

As shown in Figure 5, in the pure Chinese environment, THT algorithm takes less time than DFD algorithm and shanl algorithm. DFD algorithm takes 46.53%, 47.09%, 45.41%, 44.49%, and 46.72%, respectively. Shanl algorithm takes 83.75%, 78.76%, 72.79%, 69.87%, and 65.76%, respectively. Its time performance is better than the other two algorithms; for different test texts, the change pattern of time of HT and THT algorithms is the same, that is when the number of keywords is small, the HT algorithm uses less time, but as the number of keywords increases, the performance of THT algorithm is better than HT. The main reason for this is that, with a small number of keywords, the effect of cueing is not obvious because there is not much overlap between keywords, and the extra judgments generated by cueing lead to a slightly higher time than the HT algorithm; however, as the number of keywords increases, the effect of cueing becomes obvious, and accordingly, the time of THT is less than that of the HT algorithm. Therefore, for THT, it is most suitable for the case of a very large number of keywords. Table 4 shows that the actual number of comparisons between HT and THT is much smaller than the byte length of the text to be matched in the actual operation, which is consistent with the theoretical analysis, since Chinese is a large character set and the probability of failing to match the high byte of the first character is very high.

(3) Matching performance under a mixed environment. As shown in Figure 6, in the mixed environment, the algorithms SHENL and DFD can perform correct matching, and the time used is slightly higher than the matching time in the pure Chinese environment, because the algorithm adds the operation of distinguishing character types; SHENL algorithm cannot perform correct matching, and the five groups of Text 1 are only 51.05% and 50.20% of the correct matching times. The number of matches for Text 1 is only 51.05%, 50.20%, 51.15%, 50.57%, and 61.40%, and the number of matches for Text 3 is only 49.91%, 49.66%, 51.58%, 51.89%, and 64.32% of the correct matches; the usage time of algorithm THT shows the same pattern as that of DFD algorithm in the pure Chinese environment; in terms of clustering, the pattern of HT compared with THT is the same as that of the pure Chinese environment. The results show that the performance of THT is better than that of HT when

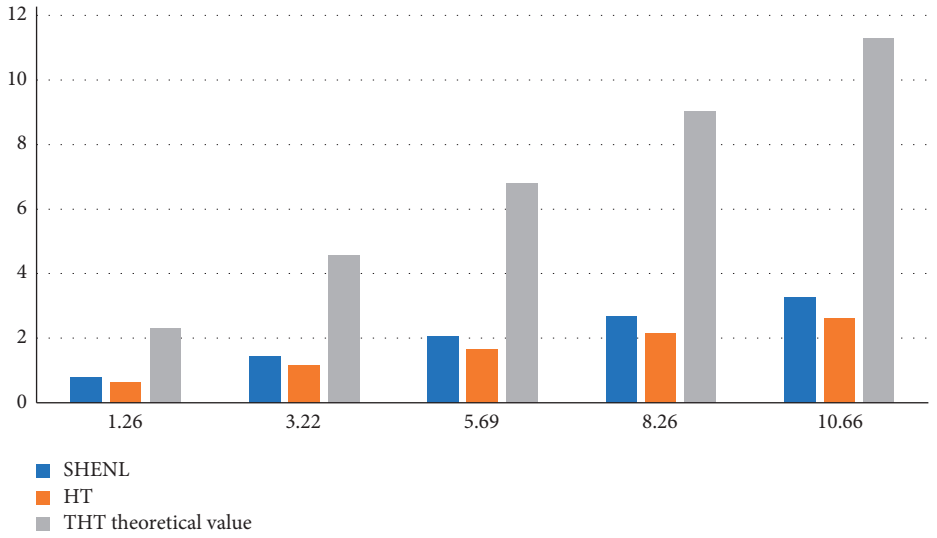

Figure 4: Comparison between various algorithms for space usage.

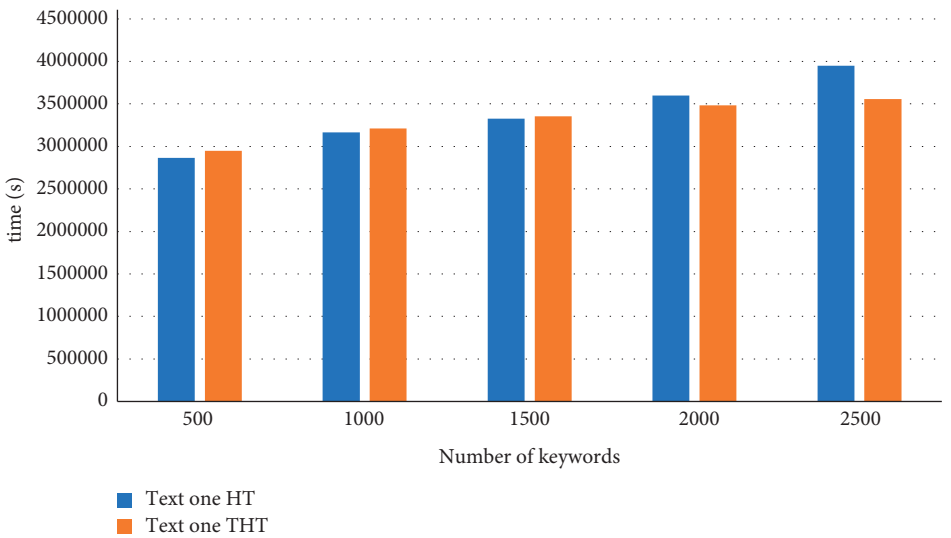

Figure 5: Comparison between algorithms for keyword matching (matching time).

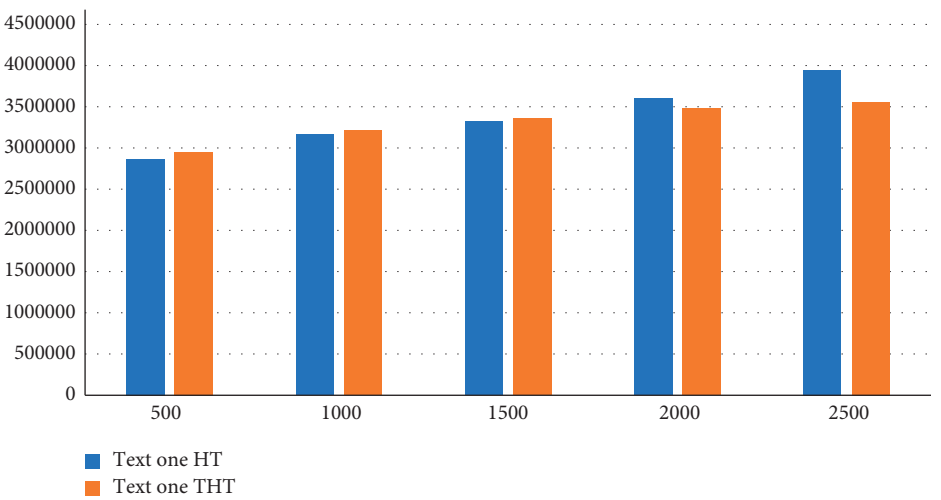

Figure 6: Comparison between the algorithms (time).

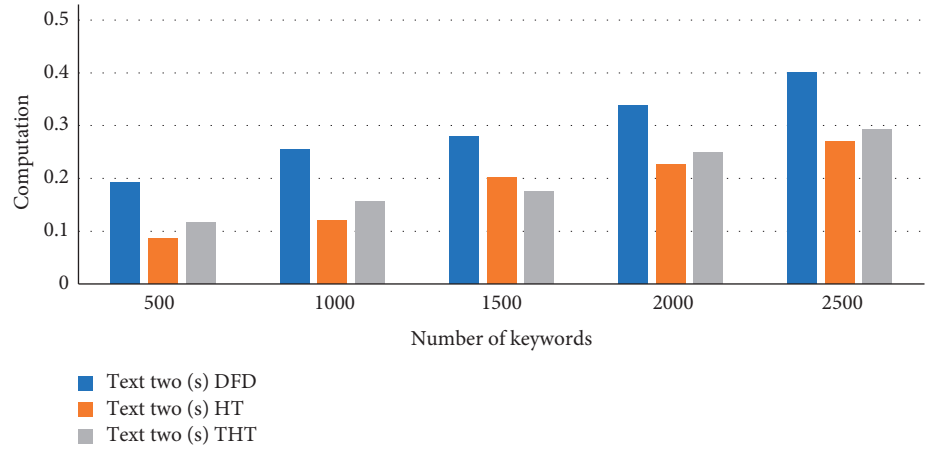

Figure 7: Computation comparison for a single text.

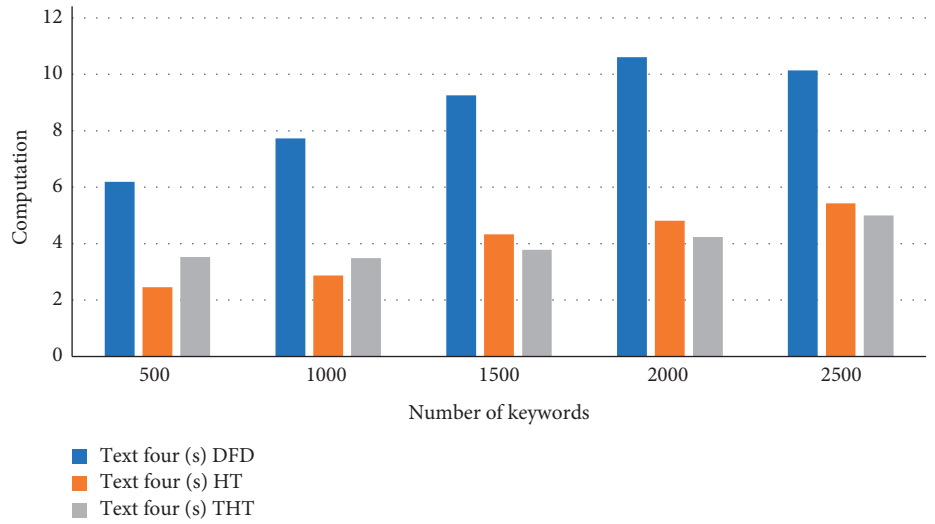

Figure 8: Comparison between the mixed text.

the keyword set is large, and it can give full play to the role of clustering. The above analysis shows that this algorithm outperforms the algorithms DFD and SHENL in terms of spatial and temporal performance and is suitable for mixed Chinese and English environments. In Figure 7, a comparison is made among all the algorithms in terms of computation for single text, while in Figure 8, a comparison is made among all the algorithms in terms of computation for mixed text.

6. Conclusion

This article proposes a novel distance-based fast multi-pattern similar string-matching algorithm for internal audit information in accounting. The algorithm uses the high and low bytes in the information as keys to construct a fully hashed trie matching machine and clusters the trie matching machine according to the characteristics of the pattern string set, so that the matching algorithm does not need to backtrack the pointer in the matching process, which effectively improves the matching efficiency and reduces the time and space complexity of the algorithm. Theoretical analysis and experimental results show that this algorithm can avoid mismatching and missing matching in the mixed information environment, and the matching speed is significantly better than existing algorithms, and there is no space expansion problem. In the follow-up work, we will further investigate the following two aspects: how to preprocess the pattern strings and determine the best order of pattern strings so that the constructed cue-complete hash trie matching machine can perform best.

Data Availability

The data underlying the results presented in the study are available within the manuscript.

Conflicts of Interest

There are no potential conflicts of interest in this article.

Acknowledgments

This work was supported by the Key Project of Teaching Reform Fund of Ningbo University of Finance and

Economics (20JYZD06): Teaching Reform of "Three-step Progressive" Practice Courses to Cultivate Students' Comprehensive Audit Ability.

References

[1] N. Tuck, T. Sherwood, B. Calder, and G. Varghese, "Deterministic memory-efficient string matching algorithms for intrusion detection," in *Proceedings of the IEEE Infocom*, vol. 4, pp. 2628–2639, Hong Kong, China, March 2004.

[2] S. Faro and S. Scafiti, "Efficient string matching based on a two-step simulation of the suffix automaton," *Implementation and Application of Automata*, in *Proceedings of the International Conference on Implementation and Application of Automata*, pp. 165–177, Bremen, Germany, July 2021.

[3] A. A. Karcioglu and H. Bulut, "The WM-q multiple exact string matching algorithm for DNA sequences," *Computers in Biology and Medicine*, vol. 136, Article ID 104656, 2021.

[4] S. Song, G. Gu, C. Ryu, S. Faro, T. Lecroq, and K. Park, "Fast algorithms for single and multiple pattern Cartesian tree matching," *Theoretical Computer Science*, vol. 849, pp. 47–63, 2021.

[5] N. Liu, F. Xie, and X. Wu, "Suffix array for multi-pattern matching with variable length wildcards," *Intelligent Data Analysis*, vol. 25, no. 2, pp. 283–303, 2021.

[6] J. P. Davis, C. Dray, N. Petrov, and E. Belanova, "Low prevalence match and mismatch detection in simultaneous face matching: influence of face recognition ability and feature focus guidance," *Attention, Perception, & Psychophysics*, vol. 83, no. 7, pp. 2937–2954, 2021.

[7] K.-F. Wong, V. Y. Lum, and W. I. Lam, "Chicon-a Chinese text manipulation language," *Software: Practice and Experience*, vol. 28, no. 7, pp. 681–701, 1998.

[8] K. F. Wong, "String matching on Chinese/English mixed texts," *Int'l Journal of Computer Processing of Chinese and Oriental Languages*, vol. 10, no. 1, pp. 115–126, 1996.

[9] F. J. Fiori, W. pakalén, and J. tarhio, "Approximate string matching with SIMD," *The Computer Journal*, vol. 81, 2021.

[10] A. A. Karcioglu and H. Bulut, "Improving hash-q exact string matching algorithm with perfect hashing for DNA sequences," *Computers in Biology and Medicine*, vol. 131, Article ID 104292, 2021.

[11] Q.-D. Sun, X. B. Huang, and Q. Wang, "Multiple pattern matching on Chinese/English mixed texts," *Journal of Software*, vol. 19, no. 3, pp. 674–686, 2008.

[12] Y. C. Wang, *The Technique and Basis of Chinese Information Processing*, Shanghai Jiao Tong University Press, Shanghai, pp. 30–31, 1990, (in Chinese).