CPAT-Tree-Based Language Models with an Application for Text Verification in Chinese

Chun-Liang Chen[1], Bo-Ren Bai[2], *Lee-Feng Chien[3] and Lin-Shan Lee[1,2,3]
[1]Dept. of Computer Science and Information Engineering, National Taiwan University
[2]Dept. of Electrical Engineering, National Taiwan University
[3]Institute of Information Science, Academia Sinica
Taipei, Taiwan, R.O.C.
E-mail: {liang, white}@speech.ee.ntu.edu.tw, {lfchien, lsl}@iis.sinica.edu.tw

Abstract

PAT tree is an efficient n-gram indexing structure. Except for text retrieval, it is believed also useful in many natural language processing applications for the construction of n-gram language models. But, an original PAT tree requires much space in memory to maintain fast speed of n-gram access and is limited to construct a large language model in practical environments. The purpose of this paper is to present an improved PAT tree structure, called CPAT tree (Compact PAT tree) for natural language modeling applications. The CPAT tree can significantly reduce the main memory requirement of original PAT trees and is found very efficient in constructing large n-gram language models. Such an advantage has been proven in OCRed-text verification and will be also introduced in this paper.

1. Introduction

PAT tree is an efficient n-gram indexing structure [Frakes 1992]. Except for text retrieval, it is also useful in many other applications, for example topic classification, spelling error detection and correction, DNA sequence search [Ricardo 1992], or even Markov n-gram language models. Markov n-gram language models were frequently used in many natural language processing applications, such as speech recognition, OCR, input methods, etc. Due to the considerations of memory space and computational complexity in practical implementation, conventional models are often an approximation, e.g., bi-gram or tri-gram models. Natural language processing systems based on such an approximated model cannot be robust enough. PAT-tree-based n-gram indexing was found very efficient to construct high-order n-gram language models in our previous work [Chien 1997]. Nevertheless, an original PAT tree requires much space to store the whole tree in the internal memory to maintain fast speed of n-gram access. It will not as efficient when the PAT tree is too large to be loaded into

memory. To run a large language model using the PAT tree indexing, it needs other improvements. The purpose of this paper is to present an improved PAT tree structure, called CPAT tree (Compact PAT tree). The CPAT tree can significantly reduce the main memory requirement of original PAT trees and is found very efficient in constructing large n-gram language models.

The CPAT tree is extended from original PAT tree. It separates a PAT tree into memory part and disk part. The memory part is basically a linear transformation of original tree structure. Pointers for tree traverse are no longer required. On the other hand, the disk part is primarily the recorded information such as string contents, frequency values etc, which are removed from main memory to release more space for allocating larger models.

The first application performed by using the CPAT trees is the experiment on OCRed-text verification in Chinese. It needs to note that the concept of "text verification" has a little difference from conventional "spelling checking". Since there is no explicit delimiters as a marker of word boundary in Chinese and some other Asian languages, the function of "text verification" in Chinese is primarily to check the validity of a text at the context level rather than word level as found in conventional English spelling checkers. The text verification problem is therefore defined to verify the validity of an arbitrary text string, including detect various input errors, e.g. speech recognition errors, typing errors, OCR errors, etc., and correct them automatically.

Primary methods for Chinese text verification (or text error checking) can be divided as dictionary lookup [Shr 1992, Liu 1997], n-gram analysis [Shr 1992, Chang 1994, Xia 1996] and parsing. The dictionary lookup approach needs to face with the word segmentation and rigid dictionary collection problems. The n-gram analysis approach, instead, relies much on the adopted bi-gram or tri-gram models. As to the parsing approach it is seldom found because it is unable to perform effective sentence parsing in Chinese texts at present. Although all of these methods can combine with morphological rules or heuristics about similarity in shapes, pronunciations, meanings or input keystrokes between similar characters for advance processing [Shr 1992, Chang 1994, Liu 1997], it is believed the text verification problem has a lot of space to improve. The CPAT-tree-based approach is proposed of this purpose. The proposed text verification process functions like spelling checking as in commercial word processors, but with high degree of differences in the used technology. Instead of detecting errors primarily at the word level as was done in conventional spelling checkers, global analysis up to the sentence level can be handled in the proposed text verification technique by means of a CPAT-tree-based large-scale language model and sophisticated text searching skills. As found in our experiments on verifying OCRed texts, the

proposed CPAT approach compared with the above methods is more competitive, if the adopted CPAT tree is trained with a sufficient corpus and more effective models continually developed.

2. CPAT Tree

PAT tree and PAT array are two well-known and frequently-used data structures for n-gram indexing in text retrieval. PAT tree is based on PATRICIA algorithm [Morrison 1968] for indexing every possible position in a continuous data stream. Each indexing point of interest is called a semi-infinite string or different suffix. PAT array [Clark 1996] is another compact representation of PAT tree. It can be considered as a sorting collection of all external nodes of PAT tree. For large text retrieval, there have been many previous researches about finding an efficient n-gram indexing data structure to take both time and space into consideration. P. Ferragina [Ferragina 1996] proposed a text indexing structure for secondary storage, which is called SB-tree, that combines the B-tree and suffix arrays. E. F. Barbosa [Barbosa 1995] proposed an optimized algorithm to improve the retrieval time of the indirect binary search in PAT array. In Sato's paper [Sato 1997], a new data structure called TS-file (Tree Structured file) and a set of algorithms were proposed to make arbitrary string retrieval especially fast. In addition, M. Shishibori [Shishibori 1997] designed a compact data structure for digital search trie and introduced a hierarchical structure in order to improve the efficiency of large registered keys retrieval. The compact concept proposed in the CPAT tree is similar to that in Shishibori's work, but the main difference is that Shishibori uses binary search tree (the terminal node stores the registered keys) while the CPAT tree proposed is a binary search trie (each node can be a terminal node or non-terminal node). Furthermore in CPAT it is added a "booster" data structure for search speedup.

2.1 PAT Tree Data Structure

The proposed CPAT tree is extended from original PAT trees. The superior features of the PAT tree data structure mostly come from its ability to perform fast full-text indexing and searching. Using this data structure to fully index the documents, all possible words or character strings, including their frequency counts in the documents, can be updated and retrieved in a very efficient way. Besides, the data stream to be indexed can be any type of information, including part-of-speech strings, phone strings or other strings depending on applications.

For convenience of description, an example data stream with two sentence fragments "個人電腦" and "人腦" is shown in Fig.1(a), in which the "Position" above the "Data

stream" means the real byte offset of the indexing points and the "Possible suffix strings" marks the 5 unique suffix strings, i.e., "個人電腦", "人電腦", "電腦", "腦", "人腦". Fig.1(b) also shows the corresponding binary bit streams of each indexed suffix string. In the processing, all of the indexed suffix strings are appended an marker "$" to identify the ending. Besides, Fig.1 (c) shows the physical representation of corresponding PAT tree, in which each node represents a unique suffix string and is associated with four-tuple of information including "comparison bit number", "frequency count", "accumulated frequency count" and "data position". The "comparison bit number" is used to indicate the bit number needs to compare and decide the left or right way to go when traversing at this node. The "frequency count" is the number of total frequency value of the indexed suffix string occurring in the data stream. The "accumulated frequency counts" stands for the sum of frequency counts of the total nodes in the sub-trees. The data position is the pointer to the data stream.

The detailed steps of the construction of PAT tree are ignored here. It is based on the process of binary search trie insertion. When a node pointed to a suffix string is inserted into a PAT tree, it will be inserted into the neighborhood of the node with a longest bit stream similarity and will be tagged with the minimal comparison bit to discriminate them. If the newly inserted suffix string has been registered in PAT tree, only the frequency counts of the representing node will increase but no extra node is needed. Just like the example in Fig.1 (c), the node C is used as a shared indexing node for pattern "腦" both in "個人電腦" and "人腦" in the corpus. The total number of nodes in PAT is exactly equal to the total number of unique or distinct suffix strings in the corpus. The advantage of the PAT tree structure is useful for speeding up searching. For illustration, the full traverse for searching for the pattern "電腦" with the pre-constructed PAT tree is demonstrated here. At first, we encode the character string "電腦" as its binary representation (BIG5 code) "1011100101110001....". The searching process will start from the root A. At the first step, the default branch to go is left because the root is a dummy node. At this time, it will stop at the node B and the comparison bit to check is bit 4. To take a look at 4^{th} bit of "電腦", it is 1. So, it takes the right branch to go. Now it will stop at node C and check the 8^{th} bit as indication. It is 1 as usual and has a right branch to go also. By the right branch, it will return back to the node B and find the comparison bit changed from 8 to 4. In the PAT tree, when the examining comparison bit is lower than the previous one, it means the branch is an upper link or the destination node is an external node. At last, by the data position pointer of node B, the destination suffix string "電腦" will be extracted from the data stream, and, after making a string comparison , it can be proven that the examining string "電腦" appears in the data stream. From the associated information, it also knows that "電腦" occurs one time and the accumulated frequency count is 6.

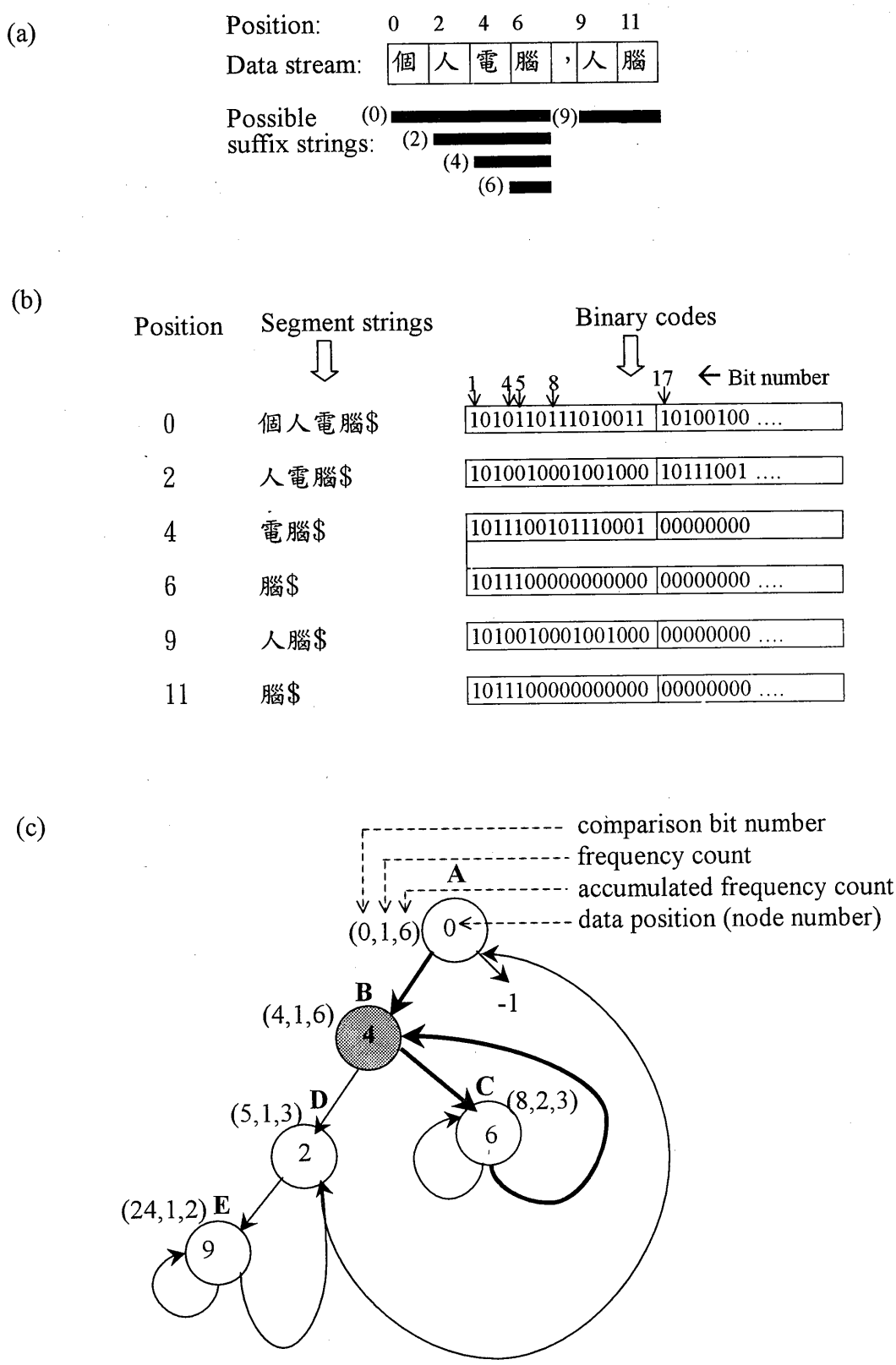


Fig.1 : PAT tree data structure

2.2 CPAT Tree Data Structure

PAT tree and PAT array are compromise in their features. The advantages of PAT tree are easy for update and fast for search. But, an original PAT tree requires much space in the internal memory to maintain fast speed of n-gram access. On the other hand, in PAT array the sorted array is usually stored in disk and indirect binary search performed for text retrieval. PAT array is flexible to index a huge data stream. But the random access of the pointers in disk slows down the searching speed. CPAT tree is developed to find out a tradeoff between PAT tree and PAT array.

Before constructing a CPAT tree, its original PAT tree must be built as a temporal media at the first stage. Then the transformation process will be performed to compress PAT tree into CPAT. The original PAT tree is separated into two parts: the memory part (RamPart) and the disk part (DiskPart) in CPAT as in Fig. 2. The RamPart is basically a linear transformation of the original tree structure. Pointers for tree traverse are no longer required. On the other hand, the DiskPart is primarily the recorded information such as string contents, frequency values etc, which are removed from main memory to release more space for allocating larger models. The whole CPAT tree looks like an iceberg. The RamPart can be viewed as the top of CPAT or iceberg. The DiskPart is that under water, which is often several times larger than the RamPart above water. This kind of indexing structures can release space in memory and afford larger or multiple indexing trees.

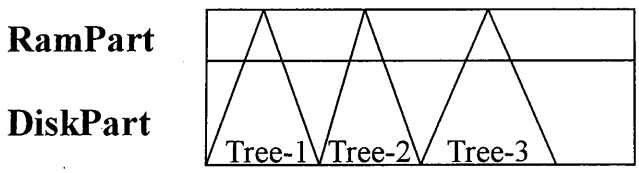


Fig. 2: Multiple CPAT trees

The RamPart should be loaded into memory before the searching starts. The RamPart consists of TreeMap and BitMap as in Fig.3. The TreeMap is a preorder mapping representation from 2-D PAT tree to one dimension of binary sequence in which '0' means internal node and '1' means external node. A node is referred to as an external node only when its comparison bit is equal to or greater than that of its parent nodes. At the same time, the BitMap is a linear array which stores the corresponding comparison bit number in sequence for each node. The stream in the BitMap will be aligned and packed with that in the TreeMap as a two-byte sequence for the sake of memory saving.

As for the DiskPart, it stores some useful information of the indexed nodes, including frequency counts, accumulated frequency counts and pointers to starting address of the indexed suffix in the data stream. Only the information of the external nodes need to save with the sequence of external nodes in TreeMap.

The left and right branches of original nodes in PAT tree have been eliminated in CPAT for saving space. The problem arising here is how to reach the left and right child for each node in TreeMap. The left child is not hard to find since in preorder sequence the left child is just the next node in TreeMap, but the right child is not so natural to examine. It should be reached based on a property of common tree structure that "the number of external nodes is exactly greater by one than that of internal nodes in a binary tree".

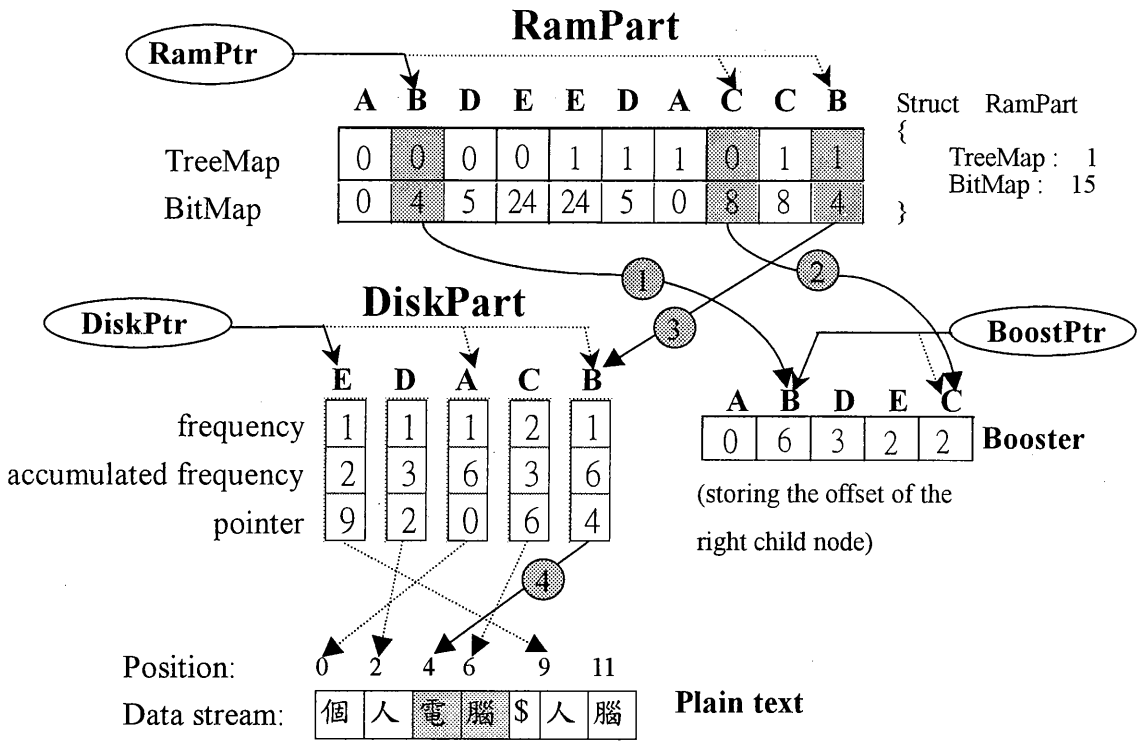


Fig. 3: Data structure content of CPAT for PAT tree of Fig. 1.

To jump from a certain node to its right child node, it should stride the overall left sub-tree . So until it jumps a sequence of bit streams in the TreeMap such that number of '0' bits is greater than that of '1' bits by one, we exactly get the right child node. Because the sequential scanning in binary bit streams will cost much CPU time, another linear array called "booster" is used here for storing the forward offset of right child node for each internal node in TreeMap. The function of "Booster" is to speedup the search.

We will use the same example for illustration with the CPAT tree. The total searching process will be divided into 4 steps with grayed and numbered circles shown in Fig.3. Besides, there are three pointers named "RamPtr","DiskPtr" and "BoostPtr" are used to indicate the current positions in the corresponding arrays during searching.

At the first step, we still ignore the dummy root node and directly go to node B (the second cell in TreeMap). So the RamPtr will point to B at this moment. According to BitMap, we check the 4th bit of binary stream of "電腦" and find it is '1'. Since node B is marked as '0' in TreeMap, it is an internal node as definition. Shift the BoostPtr to the corresponding node B and get the offset '6' for forwarding to next internal node C. At this time, the RamPtr points to node C in RamPart and pointer BoostPtr moves to next internal node C in Booster. As for the DiskPtr, it will move to node A since there are three external nodes (E, D, A) been passed from node B to node C.

Then second step, like the first step, it will check the 8th bit of binary stream of "電腦" and find it is still '1'. By the BoostPtr, the RamPtr will move forward 2 elements to reach the right child of internal node C. At the same time, it passed through 2 external nodes (C,B), so the DiskPtr also moves forward 2 elements and gets node B.

Now, at the third step, we find the element in TreeMap is marked '1', which means that the destination node has been reached. The whole record about node B will be therefore retrieved through the DiskPtr. The last step is just retrieving the string content of node B and making a comparison with the examining string "電腦".

2.3 Performance Evaluation of CPAT tree

This section shows the obtained results on both time and space tests with CPAT and PAT tree. The tests were performed under the following environment: PII-266 PC, NT workstation 4.0 and Quantum SCSI disk.

Some theoretical values of space needed by PAT tree, CPAT and PAT array are listed in Table 1 for reference. If the data stream, i.e., the Chinese text for indexing, is n bytes (n/2 characters), the theoretical space needed by PAT tree is O(9n)~O(10n), PAT array is O(5n) and CPAT is O(7n), as our observation. In this table, the RamPart size of CPAT doesn't include the "Booster" size, since it depends on if it is loaded into memory. If it is, the memory space will expand to O(2n) but n-gram access will be accelerated.

Strategy	Text Size	PAT tree	CPAT tree	PAT array
RamPart	0	9n~10n	n	0
DiskPart*	n	0	6n**	5n
Total	n	9n~10n	7n	5n

Table 1: Space usage comparison (theoretical values) in which "*" indicates that the DiskPart includes size of plain text (n), pointers (2n), counts (n) and frequencies (n), and "**" that includes booster with size n.

Then, the practical time and space needed by CPAT and PAT trees are listed in Table 2. The corpus used for testing are as follows:

Test1-O(1K) :[三字經] full text
Test2-O(10K) :[中華民國憲法] full text
Test3-O(100K) :[清靜經][道德經][金剛經][心經][六祖壇經][大學][中庸] full text
Test4-O(1M) :[紅樓夢] full text
Test5-O(10M) :[金庸小說] full text
Test6-O(20M) :[1997 中央社上半年新聞] full text
Test7-O(100M) :[1997 中央社全年新聞] full text

The obtained average ratio value (the space needed with respect to original text size as 1) in the last row in Table 2, is lower to almost 50% as compared with the theoretical value in Table 1. Although the theoretical DiskPart space is O(7n), in real test only O(3.7n) space requirement is required. This is because there exist many repeated suffix strings in the indexed data stream. The repeated suffix strings will not take space to store but only updates the frequency counts. Table 2 also shows the real time spent for constructing various sizes of PAT and CPAT trees.

CORPUS ID	SPACE (KB)				Time(sec)	
	Corpus Size	PAT	CPAT		CPAT	CPAT
			Ram	Disk	Construction	Transformation
Test1-O(1k)	3	23	4	15	0.03	0.30
Test2-O(10k)	22	122	21	79	0.27	0.13
Test3-O(100k)	111	544	100	260	1.16	0.49
Test4-O(1M)	1,791	10,839	1,885	7,069	35.02	18.13
Test5-O(10M)	12,013	64,984	11,182	42,618	272.01	413.43
Test6-O(20M)	19,541	82,779	14,587	53,604	508.87	680.93
Test7-O(100M)	107,333	439,087	107,771	397,447	2381.00	3214.35
Ratio for Test7	1	4.09	1.04	3.7	1	1.35

Table2: Time and space comparison between PAT and CPAT.

Furthermore, Table 3 shows the tests on n-gram access speed with PAT and CPAT trees. The indexed data stream is "金庸小說" (about 12MB) and the examining n-gram strings are automatically generated by keyword extraction from the data stream. The number of keywords for each n-gram is shown in Column 2. The time unit for speed measurement is second here. It is clearly to see that the time spent on CPU, hard disk and totally needed. In "Average" column, it shows the capability of how many n-grams can be accessed per second with PAT and CPAT respectively. The last column "Speed Ratio" means the ratio of CPAT over PAT in "average" column. It's obvious that disk access time always dominates the total access time. Although the achieved access speed with the CPAT tree is slower than that with the PAT tree in main memory, it was found fast enough in many natural language processing applications.

Length	# keyword	PAT	CPAT			Average		Speed Ratio
		CPU	CPU	HD	Total	CPAT	PAT	
2-gram	5047	0.170	0.400	30.405	30.805	164	29688	181.21
3-gram	2221	0.080	0.190	8.472	8.662	256	27763	108.28
4-gram	3447	0.160	0.251	7.430	7.681	449	21544	48.01
5-gram	678	0.100	0.080	1.082	1.162	583	6780	11.62
6-gram	414	0.030	0.050	0.541	0.591	701	13800	19.70
7-gram	183	0.020	0.040	0.180	0.220	832	9150	11.00
8-gram	20	0.000	0.000	0.020	0.020	1000	N/A	N/A
9-gram	7	0.000	0.000	0.010	0.010	700	N/A	N/A

Table 3 : Tests of N-gram access speed with PAT and CPAT trees.

3. OCRed-Text Verification

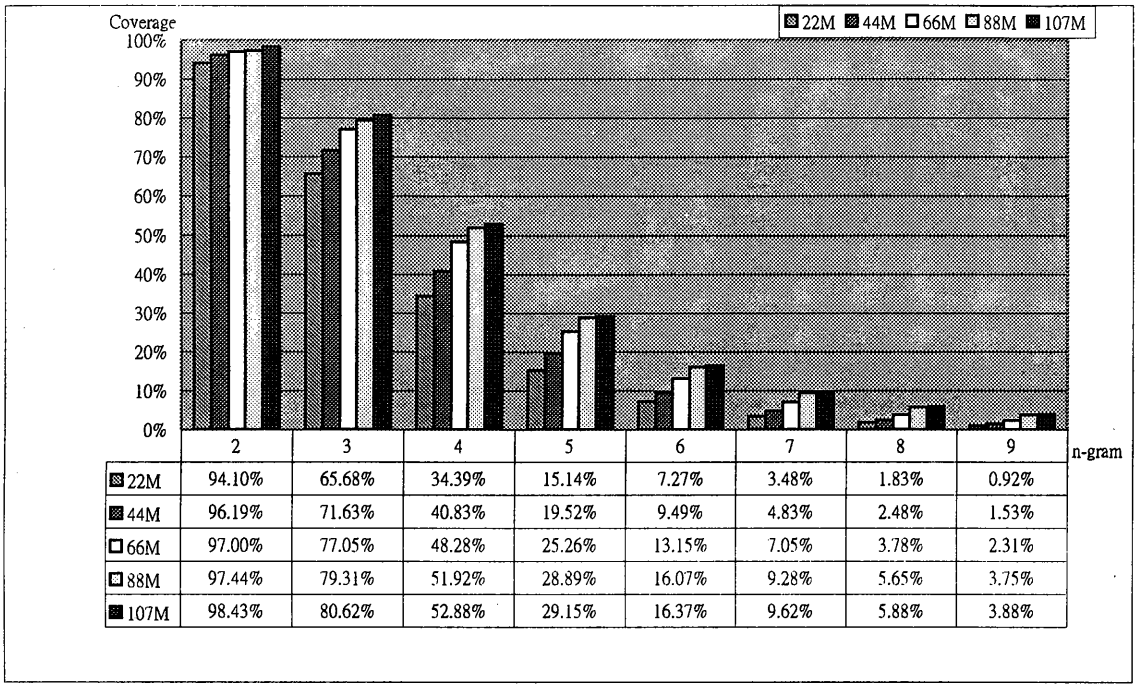
Optical Character Recognition (OCR) has been widely used for entering printed texts into computers especially for languages like Chinese, for which the complicated characters make it difficult to enter the texts through keyboards. But because such OCR processes always produce some errors, manually verifying the entered characters becomes very time-consuming. It is therefore highly desired to do such verification automatically by machine. Since it is easy to search for arbitrary character string patterns and their frequency counts in a CPAT tree, any such pattern in the entered text which has never appeared in a large text collection, or in the corresponding CPAT tree, will very possibly represent an OCR error. As a result, we can simply check the existence and frequency counts of any such character string patterns of the entered texts with the CPAT trees constructed previously to detect the errors.

This is a very attractive application of the CPAT-tree-based language models mentioned here, because errors always occur in any text entering methods, regardless of whether it is OCR, handwriting recognition, speech recognition, or even keyboards.

For Chinese and some other Asian languages this is similar to the spelling checking problem in western language, but with much higher degree of difficulties. In western language the words are well defined so simply checking the spelling of each word with a lexicon will give most of the spelling errors, but in Chinese or some other Asian languages, as mentioned before, there are no explicit word boundaries in the texts and no commonly accepted lexicon can be used in such checking processes. Therefore, instead of detecting errors primarily at the word level as was done on western languages, global analysis up to the sentence level have to be performed to handle texts without explicit word boundaries. With the approach proposed here, the full-text indexing functions given by the CPAT trees can provide the desired solution and avoid the need to use other sentence level knowledge such as grammar rules and syntactic structures. Here we'll simply use such an OCR output verification problem to test the feasibility of the proposed CPAT-tree-based language modeling techniques.

In the tests, each sentence of the OCR output is first segmented into all possible character string patterns, then each of these patterns is fed to the CPAT trees to check its existence and extract its probability (normalized frequency counts) to appear in the CPAT trees. In other words, if a sentence consists of N characters, then there are totally N*(N-1)/2 variable length patterns should be examined. In this way it is very easy and efficient to identify the character string patterns with recognition errors if it is not covered in the n-gram examining process. Actually, the power of error detection is proportional to the coverage rate of variable n-gram in the corrected testing data. The coverage rate is the percentage of total n-grams in corrected testing data that appear in the CPAT trees. The all coverage rates for n changes from 2 to 9 are shown in Fig. 4. As the corpus size increases from 22MB to 107MB, the bigram coverage rate also increases from 94.10% to 98.10% and trigram increases from 65.68% to 80.62%. Even for 5-gram, the coverage rate is approaching 30% under 107MB corpus. It is believed that the higher n-gram coverage rate, the more robust or reliable the n-gram language model is. We even can have an assumption that as the training corpus grows up to a huge size, there is no need to smoothing. If a n-gram never appears in a huge corpus, then the probability that it is an error is very large. OOV (out of Vocabulary) is another exceptive phenomenon. In addition to the statistical information extracted from the CPAT trees, some heuristic rules are found to be very helpful and integrated into the verification process as well. One example rule is that longer patterns can be considered valid if it is found to appear in the CPAT trees even with relatively lower probabilities or frequency counts, while shorter patterns need higher probabilities or frequency counts in the CPAT trees to

support its validity. This is certainly due to the fact that a longer pattern itself provides linguistic information with higher reliability.



	2	3	4	5	6	7	8	9
22M	94.10%	65.68%	34.39%	15.14%	7.27%	3.48%	1.83%	0.92%
44M	96.19%	71.63%	40.83%	19.52%	9.49%	4.83%	2.48%	1.53%
66M	97.00%	77.05%	48.28%	25.26%	13.15%	7.05%	3.78%	2.31%
88M	97.44%	79.31%	51.92%	28.89%	16.07%	9.28%	5.65%	3.75%
107M	98.43%	80.62%	52.88%	29.15%	16.37%	9.62%	5.88%	3.88%

Fig. 4: Variable n-gram coverage rates with respect to the different sizes of corpora used to construct the CPAT trees.

The testing set used here was taken from a section of printed Chinese newspaper, including 469 Chinese sentences with a total of 5,394 characters. This section of newspaper was recognized by a commercially available Chinese OCR system. The output of this OCR system is taken as the input of the OCR verification approach here. It was found manually that 131 characters among the total of 5,394 characters were incorrectly recognized. So the accuracy of the commercially available OCR system used in the test is 97.57%, and the purpose of the test here is to detect these 131 recognition errors. The results of the test are listed in Table 4 and plotted in Fig. 5 in terms of the recall rates (percentage of the 131 manually determined errors being identified correctly) and precision rates (percentage of the automatically identified errors being among the manually determined errors) with respect to different sizes of the 107MB corpora used to construct the multiple CPAT trees. In fact, the corpora used here to build the CPAT trees stem from CNA (Central News Agency) electronic news in different subject domains. It can be found from Table 4 and Fig. 5 that when the corpus size is increased from 22MB (1/5 of the whole corpus) to 107MB, the precision rate was improved significantly from 57.52% to 70.53%, while the recall rate was at the same time reduced somewhat from 67.18% to 60.30%. Such results are intuitively reasonable. A larger

corpus provides better precision performance, since more character string patterns can be observed in a larger corpus and included in the CPAT trees, thus less correct patterns will be incorrectly detected to be OCR recognition errors in the verification processes. On the other hand, when a larger corpus was included in the CPAT tree, more OCR recognition errors may be considered to be correct when the error patterns can be found as parts of some valid character string patterns in the corpus, therefore the recall rate is inevitably degraded. When measuring the system performance by the average of the precision and recall rates, the *averaged precision-recall* APR=(Precision+Recall)/2, it can be found that the APR is improved from 62.35% to 65.42% when the corpus size is increased from 22MB to 107MB. On the other hand, the same OCR testing data are also input into IBM SmartSuit'97 (a commercial word processor with Chinese text error checking functionality) to take a comparison. The results for error detection are recall 68.94%, precision 22.75% and thus APR is 45.85%.

Corpus size	22 MB	44 MB	66 MB	88 MB	107 MB
Recall(%)	67.18	64.12	61.83	61.07	60.30
Precision(%)	57.52	64.62	68.64	69.57	70.53
APR	62.35	64.37	65.24	65.32	65.42

Table 4: The error detection performance for the OCR output verification test with respect to the different sizes of corpora used to construct the CPAT trees.

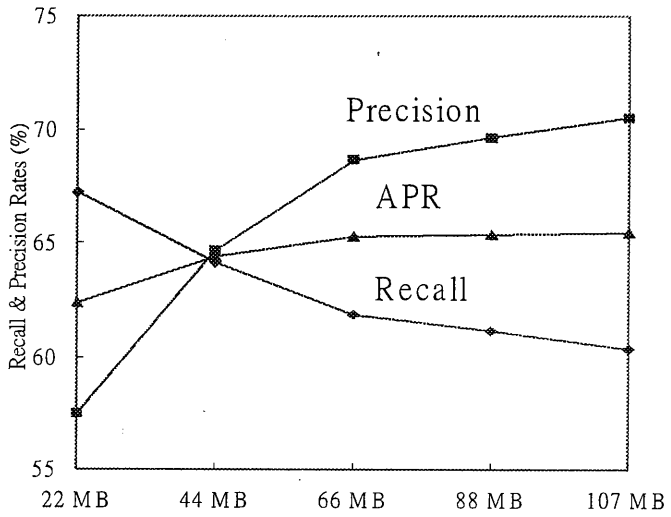


Fig. 5: The recall and precision rates for OCR output error detection with respect to the different sizes of corpora used to construct the CPAT trees.

These results are in fact significantly better then the recent results of 58% of APR obtained with a much more complicated rule based approach [Liu 1997], while in comparison here only very simple string searching techniques were used to perform the error detection. These initial results are very encouraging, and it is believed that further work will produce better performance.

4. Conclusion

The proposed CPAT data structure makes it feasible to build n-gram indexing on a large corpus and fully makes use of memory and secondary storage. It inherits both merits of PAT tree and PAT array to alleviate the memory requirement and reach a modest n-gram access speed between PAT tree and PAT array. Some initial experiments were performed to test the feasibility in OCR output verification by using a large-scale n-gram language model, which takes the CPAT tree as the core working structure. The initial result is very encouraging.

Reference

Frakes and Baeza-Yates, "Information Retrieval: Data Structures & Algorithms", Prentice-Hall, 1992.

Ricardo A. Baeza-Yates, "Text Retrieval: Theory and Practice", Information Processing 92, Vol. I, pp. 465-483,1992

L.F. Chien et al., "Internet Chinese Information Retrieval Using Unconstrained Mandarin Speech Queries Based on a Client-Server Architecture and a PAT-tree-based Language Model", Vol. 2, pp.1155-1158, ICASSP'97

Morrison, D., "PATRICIA: Practical Algorithm to Retrieve Information Coded in alphanumeric", JACM, PP.514-534,1968

Clark, D.R., and Munro, J. I. "Efficient Suffix Trees on Secondary Storage", ACM-SIAM Symposium on Discrete Algorithm ,1996

Paolo Ferragina and Roberto Grossi "Fast String Searching in Secondary Storage: Theoretical Developments and Experimental Results", ACM-SIAM Symposium on Discrete Algorithm, 1996

E. F. Barbosa, G. Navarro, R. Baeza-Yates, C. Perleberg, and N. Ziviani "Optimized binary search and text retrieval", In Algorithm- ESA'95, Third Annunal European Symposium, pp. 311-326,Greece, September 1995

T. Sato, "Fast Full Text Retrieval Using Gram Based Tree Structure", proceedings of 17-th International Conference on Computer Processing of Oriental Languages, pp.572-577, ICCPOL'97,1997

M. Shishibori, K. Morita, K. Ando and J.-I. Aoe, "The Design of a Compact Data Structure for Binary Tries", proceedings of 17-th International Conference on Computer Processing of Oriental Languages, pp.606-611, ICCPOL'97,1997

D. S. Shr et al. "A Statistical Method for Locating Typo in Chinese Sentence", Computer and Telecommunication, August pp19-26,1992

Chao-Huang Chang "A Pilot Study on Automatic Chinese Spelling Error Correction", Communication of COLIPS, Vol. 4, No 2, pp143-149, 1994

Y. Xia, X. G. Chang, S. P. Ma, X. Y. Zhu and Y. J. Jin "Co-occurrence Probability Between Chinese Characters" Communications of COLIPS, Vol. 6, No.1, pp.19-23, JUN 1996

Yuhsiang Liu, Zhili Guo, Chiching Hsu, Shauyi He and Naipo Lee, "Checking Chinese Text Errors in the Unicode Environment", 11th International Unicode Conference and Global Computing Showcase, San Jose, CA., Sep. 1997