A new technique is proposed for N-gram language model (LM) retrieval based on minimum perfect hashing (MPH). A hierarchical data structure is used to store N-gram scores in hash tables according to the order of N-grams, and a LM score is retrieved by probing the appropriate hash table slot without collision. Both integer key and character-string key based MPH functions are studied. The proposed MPH-based technique for N-gram LM lookup was evaluated on the Switchboard database and compared with the hierarchical binary search method of ISIP and the combined hash and linear search method of HTK. The proposed MPH-based technique outperformed the ISIP and HTK methods in significantly reduced LM retrieval time for bigram and trigram LMs.

1. INTRODUCTION

Many state-of-the-art large vocabulary continuous speech recognition systems use lexical trees to reduce search space. Associated with lexical trees, language model lookahead [1] is often adopted to incorporate language model (LM) probability scores into earlier levels of tree search than at leaf nodes or word endings, so as to apply tighter pruning thresholds in beam search. In LM lookahead, LM scores are applied to every node of tree copies, and therefore the scores need to be looked up very frequently. Although the overhead time of LM lookup is fully compensated by reductions in search space and decoding time, it is desirable to make the frequent lookup operation very efficient in order to minimize overall decoding time.

LM lookup methods are tied with LM data structures. In the public-domain decoding engine of HTK [2], hash table and linear lists are combined to store N-gram LM scores. For an N-gram, its history N-1-gram is first retrieved by hashing and its current word is then linearly searched. The hash function used in HTK is not perfect and a linear chaining technique is used to resolve collisions. In the public-domain decoding engine ISIP [3], N-gram LMs are stored in a hierarchical data structure, and an N-gram score is retrieved by successive binary searches through unigram, bigram,..., till N-gram.

The current work proposes a minimum perfect hashing (MPH) based technique for N-gram LM lookup. A hierarchical data structure is used to store LM scores in hash tables according to the order of N-grams, and an LM score is retrieved by probing the appropriate hash table slot without collision. Both integer key and character-string key based MPH functions were investigated. The proposed technique was evaluated on the Switchboard database [4] and compared with the HTK and ISIP methods. The proposed technique outperformed the HTK and ISIP methods in reduced LM retrieval time. This paper is organized in five sections. In section 2, the LM lookup method of ISIP is reviewed. In section 3, the proposed MPH-based technique is described. In section 4, experimental results are presented. A conclusion is made in section 5.

2. ISIP LM LOOKUP METHOD

In Fig. 1, the hierarchical data structure for N-gram LM (N = 3) in ISIP 5.8 is shown to be consisted of three levels: unigram, bigram, and trigram. Each level has a list of nodes. Each node contains five entries: current word ID, LM score, backoff score, pointer to the list of words in the next level that has the current word and its predecessor words as the history, and the size of the word list. In this way, the nodes of the nth level are clustered by their history n-1-grams. N-grams that did not occur in the training data set are not stored, and the LM data structure is therefore compact in memory usage.

At each level, words within a cluster are sorted by the word IDs. For example, when a trigram score P(w3|w1w2) is searched, w1 is first searched over the entire unigram list, w2 is next search at the bigram level in the cluster as specified by unigram node w1, and w3 is finally searched at the trigram level in the cluster as specified by the bigram node (w1w2). This process can be extended to higher-order N-grams in a similar way. Standard binary search is performed at all levels. If the trigram is found, the LM score is retrieved in the trigram node. If it is not found, a backoff is made to bigram P(w1|w2) with a normalizing factor r(w1,w2), and if the bigram is not found, a further backoff is made to unigram.

3. MPH-BASED N-GRAM DATA STRUCTURE

3.1. Basics of MPH

How to store data records so that they can be searched and retrieved efficiently is a classic problem in computer science. That is, given a set of records \( \{ R_i \} \), each consists of a unique identifier called the key \( k_i \) and a “data part” containing the desired information \( D_i \), it is desired to retrieve arbitrary
records from the set in the least amount time. In general, minimizing retrieval time can conflict with other goals such as minimizing storage space, and the selection of an appropriate storage organization and a retrieval algorithm involves trade-offs.

Define $U$ as a finite universe of keys $U = \{1…N\}$ with the cardinality of $N=|U|$. In a particular database, the actual set of keys used at a given time is $S \subseteq U$ with $n = |S|$, where typically $1 \leq n \ll O(1)$. Records are stored in a hash table $T$ having $m \geq n$ slots, indexed by $\{0,1,...,m-1\}$. Space utilization in $T$ is measured by the load factor $\alpha = n/m$. It is desirable for $T$ to be fully or almost fully used, i.e., $\alpha = 1$, and to allow unique access to its appropriate slot for any given key.

Given a key $k \in S$, the retrieval problem is either to locate a record corresponding to the key or to report that no such record exists. This is accomplished by hashing, i.e., applying a hash function $h$ to key $k$ and examining the slot in $T$ with address $h(k)$. If there are two keys $k_1, k_2 \in S$ such that $h(k_1) = h(k_2)$, then there is a collision of $k_1$ and $k_2$, and extra efforts need to be made to resolve the collision in order to retrieve either one of the records. If $h$ is a 1-1 function, then $h$ is called a perfect hash function (PHF) since no time is wasted in resolving collisions. A PHF allows retrieval of records from keys in one access, and therefore is optimal in time efficiency. For any form of hashing, optimal space utilization is attained when the hash table is fully loaded, i.e., when $\alpha = 1$. A hash function with this property is referred to as minimal hash function. The best situation, then, is to have a minimal perfect hash function (MPHF) where $\alpha = 1$ and no collision occurs.

The MPHF training algorithm as developed by R. J. Jenkins Jr. [5] is based on spanning tree and augmenting path [6] and has been successfully used for large set of records. This MPHF algorithm is adapted for use in the current problem of N-gram LM score retrieval.

### 3.2. The MPHF algorithm

In the adopted MPHF algorithm, the hash function is of the form $f(k) \oplus tab_g(k)$ or $A^\top tab[B]$, where $f(k)$ and $g(k)$ map key $k$ to an integer pair $(A, B)$, $tab[]$ is a mapping table, and “$\oplus$” is the bitwise XOR operation. The training procedure for an MPHF consists of the following three steps.

In the first step, a unique integer pair $(A, B)$ is found for each key $k$ through two functions $f()$ and $g()$, where both functions have random components to randomize the key. If the integer pair $(A, B)$ that is derived from the current key $k$ collides with an existing pair $(A', B')$ that is derived from key $k'$, i.e., $A^\top tab[B] = A'^\top tab[B']$, then the search for $(A, B)$ is restarted by using new functions $f()$ and $g()$ obtained from new random numbers or tables filled with random numbers (more details are described in section 3.4).

In the second step, the keys are grouped or segmented by their $B$ values. Each group consists of keys that share the same $B$.

In the third step, the key groups are processed by order of largest to the smallest group sizes, and the algorithms of spanning tree and augmenting paths are used to choose values for $tab[B]$. If for a certain choice of $tab[B]$, collision occurs in the form as defined in the first step, then the values in $tab[]$ are rearranged to make room for collision resolution. The goal is to fill the table so that the resulting hash values of all keys are 1-1 mapped into the range of 1 through #keys.

### 3.3. MPHF based N-gram LM data structure

The proposed MPHF based data structure for trigram, bigram and unigram is shown in Figure 2.

#### 3.3.1. String-based MPHF

An N-gram key is represented by a character string

$k = k_1k_2k_3…k_l$, where $l$ is the length of the key. For example, for the trigram $P(\text{late | he is})$, the string key is “he is late” with $l = 10$. The characters and their positions in the key are used to compute the integer pair. Considering the requirement that $f(k)$ and $g(k)$ should ensure distinct integer pair $(A,B)$ for each key, a randomization technique as proposed in [7] is used in the design of $f()$ and $g()$. The characters are coded by ASCII in the range of 0~255. Two tables filled by random numbers are used.
denoted as hashtableA and hashtableB, where the random numbers are uniformly distributed in the range of 0–2^{32}-1. At each position i and for each ASCII code k_i, there are two corresponding random numbers, one from each table. The functions f(k) and g(k) are chosen in the following form:

\[ f(k) = \sum_{i=1}^{n} \text{hashtable}_A(k_i) \& r \]
\[ g(k) = \sum_{i=1}^{n} \text{hashtable}_B(k_i) \& t \]

where & is the bitwise “and” operation, r and t are determined empirically. Two factors are considered. One is that both f(k) and tab[B] should be kept within the range of 0–#key-1. Another is that the range of g(k) should be limited as small as possible to minimize the overhead storage space for the table tab[]. The choice of

\[ r = \frac{1}{8} \cdot 2^{[\log_2 \#key]} - 1 \quad \text{and} \quad t = \frac{1}{4} \cdot 2^{[\log_2 \#key]} - 1 \]

has been proven work well for the experimental data.

During training, the functions f(k) and g(k) are computed for each key k. If for two keys k ≠ k', the integer pairs (f(k), g(k)) and (f(k'), g(k')) are identical, then the two random tables are refilled and the functions f(k) and g(k) are recomputed for all the keys. This trial-and-error process repeats until all integer pairs are distinct in the training set.

3.4.2. Integer-based MPHF

It is assumed that the vocabulary size is less than 64K, and therefore word indices are represented by short integers with 16 bits. Integer-key based MPHFs were separately designed for bigram and trigram.

For a bigram w_iw_{i+1}, the key consists of two word indices. By combining the two 16-bit integers q(w_i) and q(w_{i+1}), an intermediate 32-bit key is formed by placing q(w_i) in the high 16-bits and q(w_{i+1}) in the low 16-bits, and the resulting number is denoted as q(w_{i+1}q(w_i)). A uniformly distributed pseudorandom 32-bit number is added to q(w_{i+1}q(w_i)) and the resulting number is denoted by s. The bits of s is further mixed up to compute f(k) and g(k) by the following steps:

\[ s = ((s >> 16) \& r) + ((s << 16) + s) \]
\[ f(k) = s \& r; \quad g(k) = s \& t. \]

where << 16 and >> 16 are operations of logic left shift 16 bits and right shift 16 bits, respectively, ^ is the bitwise XOR, and r and t are the same as defined above.

For trigram, the key consists of three word indices. Unlike the bigram case, putting three 16-bit integers together requires more bits which exceeds the 32-bit integer representation in commonly used machines. Therefore, two 32-bit intermediate integers X and Y are first computed for a trigram w_iw_{i+1}w_{i+2}, where X includes two word indices q(w_i)q(w_{i+1}) like the bigram, and Y is generated by adding to q(w_{i+1}) a 32-bit random number. A more complex bit-mixing procedure is performed to make the bits of q(w_i), q(w_{i+1}) and q(w_{i+2}) interact. The computation of f(k) and g(k) from X and Y consists of the following steps:

X += Y; X ^= (Y >> 13); Y += X; Y ^= (X << 8);
X += Y; X ^= (Y >> 7); Y += X; Y ^= (X << 11);
f(k) = Y \& r; \quad g(k) = X \& t.

The uniqueness of the integer pair (f(k), g(k)) is checked for all the training keys k. If for any k ≠ k’, (f(k), g(k)) ≠ (f(k'), g(k')) , then a new random number is generated for Y and the integer pairs are recomputed for all the keys again. It is noted that the bit-mixing procedures as described here for bigram and trigram are made simple in order to keep the computation of f(k) and g(k) low so as to achieve fast LM score retrieval. In general, if the bits are not mixed up thoroughly, more training iterations may need to be carried out to resolve collisions.

Both the string-key based and the integer-key based MPHFs are proven successful for the experimented task of bigram and trigram LM score retrieval. In general, the string-key is more flexible than integer key since it readily applicable to N-grams with N greater than 3. In the case of integer key, the higher the order N of N-gram, the less straightforward to find the functions f(k) and g(k) to provide distinct (A,B)’s within the 32-bit constraint. On the other hand, the integer-based method is more efficient in computing f(k) and g(k) for bigram or trigram.

4. EXPERIMENTAL RESULTS

4.1. Data and environment

In the SWB task, the vocabulary size was 33,216 and the language model size for unigram, bigram and trigram were 33,216, 319,938 and 137,938, respectively. A test set of 43 sentences were extracted from the development test set of the WS97 subset [8] of the SWB corpus. The test set perplexities for bigram and trigram were 132.56 and 106.16, respectively.

The evaluation experiments were conducted on the platform of Dell workstation with a 1.4 GHz Pentium IV processor and 1GB memory. The operating system was Microsoft Windows 2000 Server, and the compiler was Visual C++ 6.0.

4.2. Memory space usage

The memory space usage by integer MPH based data structure and that of the ISIP is shown in Table 1. Compared with ISIP, the MPHF-based method did not use links between N-gram levels and this saving outweighed the overhead storage space for hash function tables. As the result, the MPH-based LM data structure took about only 50% of memory space compared with the ISIP LM data structure.

Table 1. Comparison on LM memory usage (Kbytes)

<table>
<thead>
<tr>
<th>Memory Cost</th>
<th>Bigram</th>
<th>Trigram</th>
</tr>
</thead>
<tbody>
<tr>
<td>LM &amp; Backoff scores</td>
<td>2,759</td>
<td>2,759</td>
</tr>
<tr>
<td>Word IDs</td>
<td>1,379</td>
<td>1,250</td>
</tr>
<tr>
<td>Arrays, pointers &amp; counters</td>
<td>4,139</td>
<td>5,755</td>
</tr>
<tr>
<td>Hash Function Table</td>
<td>-</td>
<td>512</td>
</tr>
<tr>
<td>Total</td>
<td>8,277</td>
<td>4,521</td>
</tr>
</tbody>
</table>

4.3. Evaluation on MPH training time

For a given LM, the amount of training time for a MPHF is not deterministic due to the use of random numbers in computing (A, B). Training was thus performed ten times on the same set of bigrams and trigrams by generating random numbers or random tables anew each time. Training time was measured in terms of average number of tries to find distinct (A,B)’s (#tries/(A,B)), average number of tries to fill the hash table tab[] (#tries/table), and absolute average training time. The results are summarized in Table 2. It is seen that for bigram and trigram LMs, the integer-key based MPHF took less time to train than the string-key based MPHF. For integer keys, the #tries/(A,B) increased significantly from bigram to trigram, whereas for string keys, the #tries/(A,B) decreased significantly from bigram to trigram, indicating that string-key MPHF is more feasible for higher-order N-grams.

Table 2. Average MPH training times for bigram and trigram

<table>
<thead>
<tr>
<th></th>
<th>Bigram</th>
<th>Trigram</th>
</tr>
</thead>
<tbody>
<tr>
<td>Integer</td>
<td>String</td>
<td>Integer</td>
</tr>
</tbody>
</table>

EXPERIMENTAL RESULTS
## 4.4. Evaluation on LM retrieval time

### 4.4.1. Offline evaluation

An offline LM retrieval was performed on bigram and trigram in order to evaluate retrieval efficiency separately from other factors in a decoder. The test bigram set was extracted from those hypothesized in decoding 12 sentences of the test set. The total number of hypothesized bigram was 12,813,629. The test trigram set was extracted from those hypothesized in decoding four sentences of the test set. The total number of trigram word strings was 10,000,000. Both bigram and trigram included backoffs. In Table 3, the retrieval time of the MPH methods using integer key and string key are compared with the methods of ISIP 5.7, ISIP 5.8 and HTK, where in ISIP 5.7, N-grams are searched by strings rather than integers as in 5.8. Both absolute time and ratio relative to MPH using integer key are shown. The MPH methods are significantly faster than the ISIP methods, with the integer key method the most efficient. In ISIP 5.7 and ISIP 5.8, due to the level-by-level progressive search of N-grams, retrieval of a trigram score needs to go through a search at bigram level and the search time is therefore much higher than that for bigram LM. The HTK method was significantly slower than the rest methods, due to the time spent in resolving collisions and in linear searches.

#### Table 3. Offline evaluation of LM score retrieval time.

<table>
<thead>
<tr>
<th>Method</th>
<th>Bigram</th>
<th>Trigram</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>sec.</td>
<td>ratio</td>
</tr>
<tr>
<td>MPH (integer key)</td>
<td>2.844</td>
<td>6.328</td>
</tr>
<tr>
<td>ISIP 5.7</td>
<td>29.609</td>
<td>10.41</td>
</tr>
<tr>
<td>HTK</td>
<td>85.703</td>
<td>30.13</td>
</tr>
<tr>
<td>HTK</td>
<td>85.703</td>
<td>30.13</td>
</tr>
<tr>
<td>HTK</td>
<td>85.703</td>
<td>30.13</td>
</tr>
<tr>
<td>HTK</td>
<td>85.703</td>
<td>30.13</td>
</tr>
<tr>
<td>HTK</td>
<td>85.703</td>
<td>30.13</td>
</tr>
</tbody>
</table>

### 4.4.2. Online evaluation

For online evaluation, the proposed method was compared with those of ISIPs. The HTK method was not compared since the HTK decoding engine uses a linear lexicon structure and hence does not involve language model look-ahead. In order to perform online comparative evaluations with ISIP, the proposed MPHF techniques were integrated into the ISIP 5.7 and 5.8 systems. Decoding was performed on the test set of 43 sentences and performance is summarized in Table 4 in terms of time spent in LM look-ahead (LM-LA) and in overall decoding time. The timing statistics were obtained by using the system functions of time() and clock() and were measured by times of Real Time (RT).

#### Table 4. Online evaluation of LM lookahead time and overall decoding time (RT).

<table>
<thead>
<tr>
<th>Method</th>
<th>Bigram</th>
<th>Trigram</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>LM-LA</td>
<td>Overall</td>
</tr>
<tr>
<td>MPH (integer key)</td>
<td>3.352</td>
<td>28.13</td>
</tr>
<tr>
<td>MPH (string key)</td>
<td>4.524</td>
<td>29.13</td>
</tr>
<tr>
<td>ISIP 5.8</td>
<td>5.995</td>
<td>30.805</td>
</tr>
<tr>
<td>ISIP 5.7</td>
<td>14.453</td>
<td>46.533</td>
</tr>
</tbody>
</table>

For LM look-ahead, the proposed MPHF techniques significantly outperformed the ISIP systems. The improvement factor is larger for trigram than bigram, consistent with the results of offline evaluation. For overall decoding time, the improvement factor is smaller due to various other factors involved in a decoding process. The authors have made several other improvements to the ISIP 5.7 system, where the LM lookahead time was reduced to only 6.26% of total decoding time and the total decoding time was further reduced to 6.13 times Real Time.

## 5. CONCLUSION

In this paper, MPH-based techniques are proposed for N-gram LM retrieval, with the goal of speeding up LM lookahead in large vocabulary continuous speech recognition. The proposed methods enable fast retrieval of bigram and trigram scores while taking only little extra memory space for storage of hash function tables. The training time for the MPHFs is shown reasonable. Compared with ISIP 5.8, the MPH-based techniques takes significantly less time in offline LM score retrieval and in LM lookahead time of online decoding. In addition, the MPH-based N-gram LM data structure takes less memory space due to elimination of links between N-gram levels. It is expected that for higher order N-grams, the MPH-based techniques should provide larger factors of retrieval time savings. In addition, the MPHFs can be further optimized by machine-dependant instructions for fast hashing.

## ACKNOWLEDGEMENT

This work was supported in part by the National Institutes of Health under the grant 1 R01 DC04340–01A2 and National Science Foundation under the grant NSF EIA 9911095. The authors also thank Xiaolong Li for performing offline experimental evaluation on HTK.

## REFERENCES


