A Vowel-oriented Retrieval Scheme on Minimal Perfect Hashing Searching

Shiuh-Jeng, Wang[*] **and Yau-Han Chang**
*Department of Information Management
Central Police University
Taoyuan, Taiwan 333
E-mail: sjwang@sun4.cpu.edu.tw

Abstract

Another hashing function for letter-oriented keywords based vowel-letter addressing mode is proposed in this paper. Our proposal could process a large number of keywords up to thousands in efficiency. A set of particular keywords is mechanically transformed into a hashing table in terms of the sets of offset and constant number associated with the extracted letters within the keywords. Having setup the hashing table, the retrieval for a query keyword could be immediately executed through a modular operation using a key-pair comprising of an extracted letter and a specific number featured from the keyword. The manipulated times in average to address an exact keyword is notably less than that of [8] over the result of experiments. Moreover, not only the total mount of keywords processed in our scheme is more large than some other literatures, but also the collisions occurring among the keywords mapping are reduced under our proposed algorithms.

Keywords: Perfect Hashing function, letter-oriented, collision, data retrieval

1. Introduction

A fast searching to a set of particular keywords is an important issue in now electronic data processing era. The most efficient technique adopted to carry out the scenario is the hashing function. A hashing function is an arithmetic operation that directly maps the keywords into the array indices associated with their storage space. In this manner, the data searching could be quickly executed. The applications of hashing are usually seen on the keywords searching in database, the common words filtering in natural language and the keywords search engine on WWW web site of Internet and etc. However, there will probably be a situation happening that an available storage space is occupied by more than two keywords if either the storage space is not enough or the hashing function is not well-defined. This is so-called a collision in this case. To solve the problem of collisions among the keywords mapping, a perfect hashing function in which the 'one-to-one' mapping is performed from the keyword set to the range of storage location is proposed. In general, the perfect hashing function is not difficult to find if the mapping storage space is rather sufficient. Whereas, the loading factor in performing perfect hashing might become small if most spaces are remained to be empty. Therefore, a kind of hashing, say minimal perfect hashing function (MPHF), is more worth to be explored to compact the utilization of memory space. In MPHF the keywords are mapped into its corresponding addresses with the both relations of 'one-to-one' and 'onto' without losing any memory location. Consequently, the explorations of MPHF thus become the keen research to realize the fast searching on a variety of applications.

Recall the researches of MPHF, there have been found in [1-7] to optimize the usage of storage space. In Cichelli [2], the hashing function as *h(k)=length(k) + value(the first letter of k) +value(the last letter of k)* for a keyword *k* is proposed. The method is executed by heuristic manner and a table for the associated letter-oriented is built. Nevertheless, this work is not suitable for many non-trivial sets of keys. The MPHF in [5] is then presented, in which the hashing function is expressed as *h(k)=ëC/(Dk+E)û* mod *n*, where the parameters *C*,*D* and *E* are computed by the designed algorithms and *k* is obtained from the processed keywords set. Next, Chang [1] proposed a method to translate the associated keywords into a constant *C* based upon the Chinese remainder theorem (CRT). The letter-selection keyword considered in Chang's method is retrieved by performing only a modular operation, but the retrieval of a keywords set is limited in a smaller set. In 1992, some researchers proposed multiple-function hashing schemes defined as $h(key)=(g(f_1(key))+g(f_2(key)))$ mod N [3] and $h(key)=(f_0(key)+(g(f_1(key))+g(f_2(key)))$ mod N to address an exact location for a query key, where f_0, f_1, f_2 are functions that map strings into integers, and g is an integer function. Both the schemes they proposed in [3,4] were to take into account the whole

[*] **whom correspondence**

string of keyword as key feature in the design of MPHF. In recent study, Wang et al. [8] proposed a displacement addressing method on keywords hashing. In their method, the keywords set need to be divided into several subsets of adequate size to remain its performance of decent hashing times when the keywords set is larger. Accordingly, we develop a new MPHF to address the keywords' locations in efficiency in this paper to avoid the set segmentations with proper size beforehand. Furthermore, the amount of processed keywords in our scheme is more than that of [8] and the hashing times in average is even lower than the execution in [8] over the experiment show.

The rest sections are organized as follows. Sec. 2 describes our efficient vowel-oriented MPHF. Then an example of more keywords is illustrated and a comparative experiment for thousands of keywords is shown in Sec 3. Finally, the conclusions are given in Sec. 4.

2. A Vowel-oriented Retrieval Scheme on MPHF Searching

Without loss generality, each keyword considered in our scheme is assumed to be a non-trivial string of characters in English. The MPHF explored in our scheme is a way that features a key-pair of *(EL, SN)* in a keyword, where *EL* is a letter extracted from the heading letter and *SN* is a specific number computed from the distribution of the vowel-letters and the last consonant-letter within the keyword. These key-pairs, afterwards, are grouped into a table, named group table, in lexical ordering of the *EL*'s. The second components *SN*'s of key-pairs in each group would be then gathered into a representative constant. Meanwhile, the offset number of *EL* in each group is required to be recorded in order to efficiently utilize the storage space. Eventually, an addressing hashing table is thus constructed in terms of the sets of offset and representative constant.

In the following, a simple example using reserved words in PASCAL is introduced to illustrate the group table generation during hashing process so as to facilitate readers to understand our proposed algorithms later. First of all, we define the letter-to-number translation relations, R_v and R_c, for the vowel-letter and consonant-letter in English, respectively. The set of vowel-letters $\{A,E,I,O,U\}$ is then translated to a set of a sequence of numbers as $\{R_v(A)=0, R_v(E)=1, R_v(I)=2, R_v(O)=3, R_v(U)=4\}$. Follow this principle, the set of consonant-letter $\{B,C,\ldots,Y, Z\}$ is also mapped to a numerical set of $\{R_c(B)=2, R_c(C)=3, \ldots R_c(Y)=25, R_c(Z)=26\}$, where R_c(*'vowel-letter'*) is undefined. Second, the key-pair of *(EL, SN)* is characterized from each keyword. Third, group the keywords set in lexical ordering of the heading letter *EL*'s of the processed keywords. Subsequently, consider a reserved word "PACKED" in PASCAL, for instance, in which the letter-vowels and the last consonant as '*A*', '*E*' and '*D*' are posted at locations 2, 5 and 6. For the three letters, which could be encoded into an initial number *IN*= 205164 of resulting from the concatenation of the three pairs of *(L,R(L))s* =(2,0), (5,1), (6,4), where *L* denote the position number of placing a letter numerated from left-to-right starting at 1 on the processed keyword and $R(\cdot)$ is the letter-to-number translation relation for *L*. Furthermore, in order to conduct the specific number *SN*, a modulus of prime 29 is chosen to gain *SN*=*IN* mod 29=205164 mod 29=18. A pair of *(EL,SN)*= (P,18) associated with "PACKED" is therefore featured. Similarly, we could obtain the other key-pairs such as (B,23) for "BEGIN" and (E,3) for "END". Eventually, group all key-pairs *(EL, SN)s* associated the reserved words in PASCAL in the lexical ordering of the *EL*'s. The group table is then shown as Table 1 below.

Observe the Table 1, we find out that the two keywords, "THEN" and "TO" in 'T'-head group generate the same specific number *SN=7*, i.e. they could not to be distinguished in a hashing process. Therefore, a cyclic extraction process that generates next key-pair of *(EL, SN)* for the two collided words is needed to be launched until all keywords could be completely recognized. In our scheme, the next key-pairs of *(EL,SN)s*=(O,13), (H,19) are capable of being featured by the rest keywords "O" and "HEN", respectively resulted from removing the head-letter of the original words. The two words collided to each other in 'T'-head group in the first hashing would be further subdivided into a subgroup of 'O'-head and a subgroup of 'H'-head within the area of 'T'-head group for next hashing use. In conclusion, the work to thoroughly distinguish all key-pairs for the reserved words in PASCAL is done.

In order to further form the addressing hashing table, the CRT is employed to generate the representative constant planted inside the table for later keyword retrieval use. The application for CRT to generate the constant is now shown as follows:

Theorem 1 *(Chinese remainder theorem)*
Let $r_1, r_2, \ldots, r_n$ *be integers. There exists an integer C such that* $r_1=C \pmod{p_1}$, $r_2=C \pmod{p_2}$, …, *and* $r_n=C \pmod{p_n}$, *if* p_i *and* p_j *are relatively prime for all* $i \neq j$.

Theorem 2 *Let* p_i *and* p_j *be relatively prime number, where* $i \neq j$ *and* $1 \leq i,j \leq n$. *Let* $p_1<p_2<\ldots<p_n$. *Then* $C = \sum_{i=1}^{n}(b_i M_i i) \bmod \prod_{i=1}^{n} p_i$ *be the smallest positive integer such that* $C \equiv i \pmod{p_i}$ if $M_i = \prod_{j \neq i} p_j$ *and* b_i *satisfies the congruence* $M_i b_i \equiv 1 \pmod{p_i}$.

According to the *Theorem 1* and *2* mentioned above, the numbers p_i's are required to be relatively prime to each other so as to construct a constant satisfying the congruence relations. Therefore, a prime translation table shown in Table 2 is built to guarantee the p_is' conditions in CRT.

To fit for our scheme, the expression of $C \equiv i \pmod{p_i}$ in Theorem 2 is required to be adjusted as

$$RC \equiv i \pmod{p(SN_i)}$$

(1) As a result, the generation for RC is changed as the form

$$RC = \sum_{i=1}^{n}\left(b_i\left(\prod_{j\neq i} p(SN_j)\right) i\right) \bmod \prod_{i=1}^{n} p(SN_i) \text{ ,}$$

(2)where $\prod_{j\neq i} p(SN_j) \bullet b_i \equiv 1 \pmod{p(SN_i)}$.

Come after the distinct key-pair *(EL, SN)*'s featured from the particular keywords, the *RC* is constructed by CRT mentioned above. Then an addressing hashing table is built instead of the particular set of keywords for retrieval use. On the way of retrieval of a keyword, the MPHF is set as the expression:

$$H_t(EL,SN)=O_t(EL)+(RC_t(EL) \bmod p(SN)),$$

(3)where the numbers O_t's and *RCt*'s for *t³1* are key parameters to address each keyword.

The details to generate the addressing hashing table are summarized the following algorithms.

Algorithm 1: The basic group divisions that associate with the heading letter *HEL*'s in the particular keywords set.

<u>Input:</u> A set of particular keywords with the heading letter *HEL*'s, say *PKS*.

<u>Output:</u> The sets of non-integers of offset O_t's and representative constant RC_t 's that associate with *HEL*'s in a hashing *t=1*.

<u>Step 1:</u> Set the hashing time *t=1*, and group all the input keywords denoted by $G_1^{(t)}$, $G_2^{(t)}$,…$G_i^{(t)}$,…, $G_z^{(t)}$ with their heading letter HEL_i in lexical ordering.

<u>Step 2:</u> Compute the initial number *IN* for each processed keyword, where *IN* = concatenating the pairs of *(L,R(L))* for all the vowel-letters and the last consonant-letter within a processed keyword. The components *L*'s and *R(L)*'s have been defined on the previous paragraph.

<u>Step 3:</u> Compute the specific number *SN* for the processed keyword as *SN=IN* mod 29.

<u>Step 4:</u> Compare all the *SN*'s in the HEL_i-head group. Mark the keywords in which the generating key-pairs (EL_i, SN)'s have the same component *SN*, where $EL_i = HEL_i$.

<u>Step 5:</u> Translate all the *SN*'s for each group to their corresponding primes *p(SN)*'s by using Table 2.

<u>Step 6:</u> Construct a representative constant for all keywords for HEL_i-head group by (2), where the formula in (1) would be modified into two parts containing $RC(EL_i) \equiv 0 \pmod{p(SN)}$ for the *p(SN)*'s generated from marked keywords and $RC(EL_i) \equiv i \pmod{p(SN)}$, *i³1* for the unmark keywords with their associated *p(SN)*'s.

<u>Step 7:</u> Count the total number of all unmark keywords in HEL_i-head group. Then compute $O_t(EL_i)$ as $O_t(EL_i)$= $\sum_{x=1}^{i-1}\left|G_x^{(t)}\right|$, where $|G|$ denotes the cardinality of a group set containing unmark keywords.

<u>Step 8:</u> Compute $T^{(1)}$= $\sum_{x=1}^{z}\left|G_x^{(1)}\right|$.

<u>Step 9:</u> Set *t=t+1* and cut off the first letter *HEL* of the marked keywords in each *HEL*-head group to be a new processed keywords set, say $HEL\text{-}NPKS^{(t)}$.

<u>Step 10:</u> Numerate the $HEL\text{-}NPKS^{(t)}$ sets as $NPKS_1^{(t)}$, $NPKS_2^{(t)}$, …, $NPKS_s^{(t)}$, where s = the total amount of the $HEL\text{-}NPKS^{(t)}$ sets.

<u>Step 11:</u> Output the sets containing the non-negative integers offset $O_1(EL_i)$' s and $RC_1((EL_i)$' s.

Algorithm 2: The construction of an addressing hashing table.

<u>Input:</u> The set $HEL\text{-}NPKS^{(t)}$ in the *HEL*-head group.

<u>Output:</u> The sets of non-integers of offset O_t's and representative constant RC_t that associate with the extracted letter *EL* generated in hashing *t³2*.

<u>Step 1:</u> Group the $HEL\text{-}NPKS^{(t)}$ with their head-letter in lexical ordering denoted by $SG_{HEL,1}^{(t)}$, $SG_{HEL,2}^{(t)}$,…, $SG_{HEL,j}^{(t)}$,…,$SG_{HEL,r}^{(t)}$ for *t³2* and extract EL_j from the head-letter of processed keyword in $SG_{HEL,j}^{(t)}$.

<u>Step2:</u> Compute the initial number *IN* for $HEL\text{-}NPKS^{(t)}$ in Step 2 of Algorithm 1.

<u>Step 3:</u> Compute the specific number *SN* as *SN=IN* mod 29.

<u>Step 4:</u> Compare all *SN*'s in the EL_j-head subgroup. Mark the keywords in which the associated key-pairs have with the same *SN*.

<u>Step 5:</u> Translate all *SN's* to their corresponding primes *p(SN)*'s using Table 2.

<u>Step 6:</u> Construct a representative constant for EL_j-head group in Step 6 of Algorithm 1.

<u>Step 7:</u> Let $HEL\text{-}NPKS^{(t)}$ be the i^{th} set among the all $NPKS^{(t)}$ sets, *1£i£s*. Count the total number of all unmark keywords in EL_j-head subgroup. Then $O(EL_j)_t$ is computed as

$$O(EL_j)_t = \sum_{y=1}^{t-1} T^{(y)} + \sum_{NPKS_{x=1}^{(t)}}^{i-1} \sum_{y=1}^{r} \left|SG_{HEL,y}^{(t)}\right| + \sum_{\substack{NPKS_i \\ y=1}}^{j-1} \left|SG_{HEL,y}^{(t)}\right|,$$

(4)where $|SG|$ denotes the cardinality of a subgroup of keywords set.

Step 8: Compute $T^{(t)} = \sum_{NPKS_{x=1}^{(t)}}^{s} \sum_{y=1}^{r} \left|SG_{HEL,y}^{(t)}\right|$

Step 9: Set $t=t+1$. Cut off the first letter EL_j of the marked keywords in the each EL_j-head subgroup to be a new processed keywords $HEL\text{-}NPKS^{(t)}$.

Step 10: Numerate the $HEL\text{-}NPKS^{(t)}$ sets as $NPKS_1^{(t)}$, $NPKS_2^{(t)}$, …, $NPKS_s^{(t)}$, where s = the total amount of the $HEL\text{-}NPKS^{(t)}$ sets.

Step 11: Go to Step 1 to reiterate until all $HEL\text{-}NPKS^{(t)}$=Æ .

Step 12: Output O_t's and RC_t 's associated with the extracted letter EL_j for t³$2$.

After executing Algorithm 1 and 2, an addressing hashing table is constructed in terms of the sets of O_t's and RC_t 's for for t³$1$. Next, continue to consider the reserved words in PASCAL again to illustrate the setup of the addressing hashing table according to the proposed algorithms. Inspect The 'F'-head group of {FOR, FUNCTION, FILE}, for instance, following the Algorithm 1 the key-pair of *(EL, SN)*'s are first featured as {(F,2), (F,14),(F,18)}. A representative constant RC_1 is then constructed based on the CRT satisfying the congruence relations as

$$RC_1\,(F) = 1 \pmod{p(2)},$$
$$RC_1\,(F) = 2 \pmod{p(14)},$$
$$RC_1\,(F) = 3 \bmod (p(18)),$$

where $p(\cdot)$ is a prime translation shown in Table 2. Consequently, $RC_1\,(F)$=1894 is summed up by using (2). Besides, $O_1(F)$ is filled with *10* in the hashing table since it is counted from the total number of distinct key-pairs from the 'A'-head group to 'E'-head group. Then look at the marked keywords "THEN" and "TO" in the first hashing. Due to the *(EL, SN)*'s are the same, the second hashing process is thus launched inside the 'T'- head group. According to the Algorithm 2, $RC_2(H)$=108 and $RC_2(O)$=42 are generated, respectively for the rest keywords "HEN" and "O" when the first letter 'T' is cut off from the original keywords. Meanwhile, the offset $O_2(H)$=31 +0=31 and $O_2(O)$=31+1=32 are also counted out. Lastly, the addressing hashing for reserved words in PASCAL is shown in the following Table 3.

Having set the hashing table, the retrieval algorithm to address a query keyword is presented as follows:

```
Algorithm 3: Addressing an input keyword
Input: A query keyword K with heading letter HEL
Output: The address of K translated from the addressing hashing table
Step 1: Set the hashing time $t=1$.
Step 2: Feature a key-pair $(EL, SN)$ in
        $HEL$-head group for $K$ by using
        the Step 2 and 3 of the Algorithm 1,
        where the first component $EL$ is the head-letter of the processed keyword.
Step 3: Determine the computation as
        $D=RC_t(EL) \bmod p(SN)$,
        IF $D$¹$0$ THEN
            perform the Equation (3) in $HEL$-head group and go to Step 4
        ELSE
            set $t=t+1$ and cut off the first letter of the processed keyword to be a new keyword, then go to Step 2 to feature next key-pair $(EL, SN)$.
Setp 4: Output the mapping address of $K$, resulted from $H_t(EL,SN)$.
```

3. Experiments and Discussions

In this Sec., we further show an example with more keywords of VAL containing 59 reserved words to demonstrate our approach.

Example 3.1. Consider the 59 reserved words in VAL. Through the Algorithm 1 in our scheme, a group table, Table 4, is generated as shown in the following.

In Table 4, There two groups which exist the same key-pairs have been found in 'N'-head group and 'T'-head group, respectively. The RC_1 (T) in 'T'-letter group, for instance, is generated as the form as

$$RC_1\,(T) = 1 \pmod{p(4)},$$
$$RC_1\,(T) = 0 \pmod{p(7)},$$
$$RC_1\,(T) = 0 \pmod{p(20)},$$

such that RC_1 (T) =6035 and then to be stored into the addressing hashing table. Examine the marked keywords "TAG", "THEN", "TRUE", "TYPE" that need to be reprocessed by the Algorithm 2. Afterwards, the four marked keywords are further subdivided into four subgroups, cited by 'A'-head subgroup, 'H'-head subgroup, "R"-subgroup and "Y"-subgroup, inside the area of 'T'-head group of hashing table. Subsequently, the offsets and representative constants as $O_2(A)=55$, $RC_2(A)=80$, $O_2\,(H)=56$, $RC_2(H)=38$, $O_2(R)=57$, $RC_2(R)=68$ and $O_2(Y)=58$, $RC_2(Y)=38$ associated with the four reprocessed keywords are computed. Ultimately, the hashing table construction is stopped on the second process since all the 59 keywords have been come out 59 distinct key-pairs within the two hashing processes. To address the validation of hashing table, consider the keyword "TRUE" now, then the Algorithm 3 is launched. The first key-pair (T,20) is featured, then the $O_1(T)$ and $RC_1(T)$ are revealed to compute the D=6035 mod $p(20)$=0. Clearly, it is necessary to enter the second hashing process since D=0. Hence the second key-pair *(R,19)* of 'R'-head subgroup inside the 'T'-head group is obtained again. Compute D= 68 mod $p(19)$=1, so that $H_2(R,19)=O_2(R)+D$=57+1=58 in (3) is evaluated , which is the address of "TRUE". The whole hashing table for reserved words in VAL is shown as Table 5 below.

Compare to Wang et al.' s scheme [8] in which the cyclic letter-oriented based on the displacement addressing technique was proposed. Although a rather large amount keywords could be processed in their scheme, but the keywords need to divide an adequate size of keywords set in order to reduce the hashing times in querying a keyword. The reason is that all the collision keywords in each hashing process are gathered together to regenerate next key-pairs and then refilled into the hashing table in terms of non-negative integers. The manipulation for the rehashed keywords is too complicated so that the processed keywords are required to be divided beforehand. While in our scheme, not only the division for a larger keywords set is released, but also the less hashing times is gained, as observed from the result of the experiment show. That is to say, thousands of keywords could be directly translated into an addressing hashing table. Without loss the generality, the stored integer in hashing table is still large, but the most large integer happening in our scheme is limited a value of $\prod_{i=0}^{28} p(i)$, where $p(i)$ is derived from the Table 2. In practice, these digit numbers were usually stored in the form of character-string to avoid the truncation errors in data storage and then segmentalized systematically to achieve the arithmetic operation. Accordingly, the implement of the number-store hashing table is viable in real applications. In our scheme, we further perform an experiment in which there are two thousands commonly used keywords in an ordinary diction of English to illustrate the validity of our algorithms. The experiment shown in Fig 1 apparently explains that the hashing times is proportional to the set of keywords and the curve plotted in our scheme is lower than the curve shown in [8]. However, each keyword in our experiment is uniformly distributed between 2 and 20 in length. Having set up the hashing table, a simple arithmetic modular operation is required when a keyword query requests. Although the keywords set increase dramatically, the hashing times are still kept growth slowly, as observed from Fig. 1. Therefore, our scheme indeed speeds up the searching time of a hashing keyword and improves the performance of algorithms proposed in [8] on more large keywords set.

4. Conclusions

In this paper, we have proposed a new minimal perfect hashing function to implement the fast letter-oriented string searching. A key-pair of *(EL,SN)* is uniquely featured from each particular keyword during the hashing processes. Eventually, the hashing table in which two set of offsets and representative constants are planted for addressing the keyword is built to utilize in later retrieval of a query keyword. The area of each heading letter group is further divided into various subgroups to accommodate the marked keywords associated with the same the key-pair in the hashing table construction. Afterwards, each marked keyword appearing in the same heading letter group is cyclically cut off the heading letter of the rest keyword itself to gain a new key-pair for rehashing use. Consequently, our scheme could process a larger amount of keywords up to thousands. It' s also apparently observed that the hashing times executed in average is notably less than that of [8] over the result of experiment. Besides, the key-pair featured on each keyword in our scheme is gained by the strategy of cyclic extraction for the letter-oriented keyword so that the intractable letter-selection on keyword in some other literatures is avoided. In conclusion, a fast searching using MPHF for a large set of keywords is efficiently achieved in our scheme.

References

[1] C.C. Chang, "The study of an ordered minimal perfect hashing scheme," Comm. ACM. 27(4), 1984, 384-387.

[2] R.J. Cichelli, "Minimal perfect hash functions made simple," Comm. ACM. 23(1), 1980, 17-19.

[3] Z.J. Czech, G. Havas and B.S. Majewski, "An optimal algorithm for generating minimal perfect hash functions," Information Processing Letters 43(5), 1992, pp.257-264.

[4] E.A. Fox, L.S. Heath, Q.F. Chen and A.M. Daoud, "Practical minimal perfect hash functions for large databases," Comm. ACM 35(1), 1992, pp. 105-121.

[5] G. Jaeschke, "reciprocal hashing: A method for generating minimal perfect hashing functions," Comm. ACM 24(12), 1981, pp. 829-833.

[6] B. Jenkins, "Algorithm alley: Hash functions," Dr. Dobb' s J. 1997, pp.107-109, pp. 115-16.

[7] T.G. Lewis and C.R. Cook, "Hashing for dynamic and static internal tables," IEEE Computers, 1988, pp. 45-46.

[8] S.J. Wang and J.K. Jan, "A displacement addressing method for letter-oriented keys," The Journal of Systems and Software 46, 1999, pp. 77-88.

Table 1. Consider the group table of keyword set of PASCAL reserved words

Group	Keywords	*IN*	*SN*	Group	Keywords	*IN*	*SN*
1(A)	ARRAY	1040525	5	12(O)	OF	1326	21
	AND	1034	19		OR	13218	23
2(B)	BEGIN	2142514	23	13(P)	PACKED	205164	18
3(C)	CONST	23520	1		PROGRAM	3360713	19
	CASE	2041319	9		PROCEDURE	33517491818	27
4(D)	DOWNTO	2363520	20	14(R)	REPEAT	214150620	4
	DIV	22322	21		RECOED	21435164	17
	DO	2314	23	15(S)	SET	21320	5
5(E)	END	1134	3	16(T)	*THEN	*31414	*7
	ELSE	1141319	24		*TO	*23120	*7
6(F)	FOR	23318	2		TYPE	41316	20
	FUNCTION	246273814	14	17(U)	UNTIL	1442512	23
	FILE	2241312	18	18(V)	VAR	20318	18
7(G)	GOTO	2343320	4	19(W)	WITH	2248	15
8(I)	IN	12214	5		WHILE	3251412	19
	IF	1226	8				

The asterisk ‘ *’ denotes the associated word collides with other words in PASCAL

Table 2: The prime translation table

SN_i	$p(SN_i)$	SN_i	$p(SN_i)$	SN_i	$p(SN_i)$
1	2	11	31	21	73
2	3	12	37	22	79
3	5	13	41	23	83
4	7	14	43	24	89
5	11	15	47	25	97
6	13	16	53	26	101
7	17	17	59	27	103
8	19	18	61	28	107
9	23	19	67	0	109
10	29	20	71		

SN_i : the specific number computed from a keyword
$p(SN_i)$: the corresponding prime of SN under the prime translation mapping

Table 3. Address Hashing table for PASCAL reserved words

EL(HEL)	Offset: $O_1(EL)$	$RC_1(EL)$	*EL*	Offset: $O_2(EL)$	$RC_1(EL)$
A	0	672			
B	2	84			
C	3	25			
D	5	28472			
E	8	91			
F	10	1894			
G	13	8			
I	14	78			
K		18			
L	16	44			
M	17	155			
N	18	4235			
O	20	160065			
P	22				
Q		120			
R	25	12			
S	27	782			
T	29	84	A		
			H	32	108
			O	33	42
U	30	62			
V	31	471			

Table 4. A heading letter group table for VAL reserved words

Group (‘*HEL*’-head)	Translated address	The key-pair *(EL,SN)*’s	Keywords
1(A)	1	(A,1)	ADDL
	2	(A,0)	ARITHERROR
	3	(A,24)	ABS
	4	(A,5)	ARRAY
	11	(C,23)	CONSTRUCT
4(D)	12	(D,23)	DO
5(E)	13	(E,6)	EXP
	14	(E,12)	EMPTY
	15	(E,25)	ELSEIF
	16	(E,24)	ELSE
	17	(E,3)	END
	18	(E,21)	EVAL
	19	(E,19)	ERROR

6(F)	20	(F,17)	FALSE
	21	(F,22)	FORALL
	22	(F,14)	FUNCTION
	23	(F,2)	FOR
7(H)	24	(H,15)	HIGH
8(I)	25	(I,8)	IF
	26	(I,10)	IS
	27	(I,20)	ITER
	28	(I,24)	INT
9(J)	29	(J,2)	JOIN
10(L)	30	(L,7)	LOW
	31	(L,5)	LET
11(M)	32	(M,1)	MAKE
	33	(M,13)	MIN
	34	(M,24)	MAX
	35	(M,14)	MOD
	36	(M,15)	MISSELT
12(N)	54	*(N,11)	*NEGOVER
	55	*(N,11)	*NIL
	37	(N,27)	NEGUNDER
	38	(N,23)	NULL
13(O)	39	(O,4)	OTHERWISE
	40	(O,21)	ONEOF
14(P)	41	(P,16)	POSOVER
	42	(P,3)	POSUNDER
15(R)	43	(R,2)	REMH
	44	(R,9)	REPLACE
	45	(R,24)	RETURNS
	46	(R,10)	REML
	47	(R,14)	REAL
	48	(R,12)	RESULT
	49	(R,25)	RECORD
	5	(A,4)	ADDH
	6	(A,12)	AT
2(B)	7	(B,23)	BEGIN
	8	(B,14)	BOOL
	9	(B,4)	BOUND
3(C)	10	(C,26)	CHAR
16(S)	50	(S,3)	SIZE
17(T)	51	(T,4)	TAGCASE
	56	*(T,7)	*TAG
	57	*(T,7)	*THEN
	58	*(T,20)	*TRUE
	59	*(T,20)	*TYPE
18(U)	52	(U,26)	UNDEF
19(Z)	53	(Z,18)	ZERODIVIDE

The marked "*" keywords denote that the *SN* is the same in the ' HEL'-head group, and then need to rehashed at next run.

Table 5. A hashing table for VAL reserved words

EL	*Offset: O₁(EL)*	*RC₁(EL)*	*EL₂*	*Offset:O₂(EL)*	*RC₂(EL)*
A	0	32195222			
B	6	5979			
C	9	2325			
D	11	84			
E	12	84918022126			
F	19	335359			
H	23	48			
I	24	1086516			
J	28	4			
L	29	155			
M	31	5092407			
N	36	208165	A		
			E	53	44
			I	54	6
O	38	148			
P	40	161			
R	42	23978219005			
S	49	6			
T	50	6035	A	55	80
			H	56	38
			R	57	68
			Z	58	38
U	51	102			
Z	52	62			

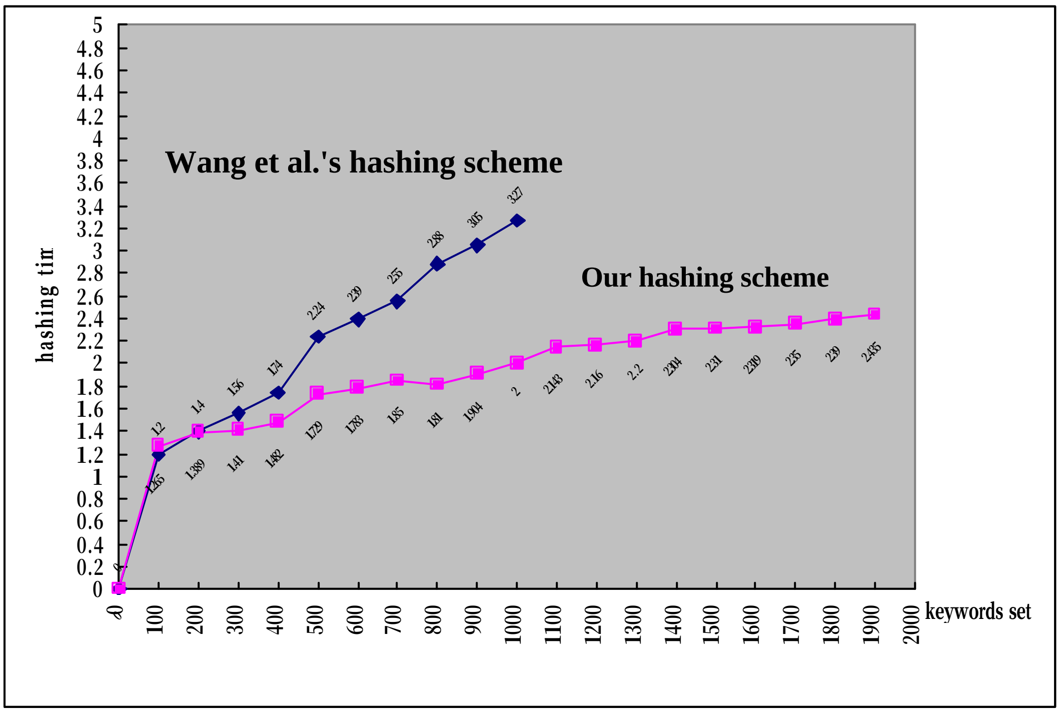

Fig.1. The experiment shows the hashing times in average between our scheme and Wang et al.’ s scheme.