Latest Trends on Computers, Vol.2, pp.483-489, ISBN 978-960-474-213-4, ISSN 1792-4251, CSCC conference, Corfu, Greece, 2010

New Hash Function Construction for Textual and Geometric Data Retrieval

Václav Skala, Jan Hrádek, Martin Kuchař
University of West Bohemia
Department of Computer Science and Engineering
CZ 306 14 Plzen, Czech Republic

skala@kiv.zcu.cz

ABSTRACT

Techniques based on hashing are heavily used in many applications, e.g. information retrieval, geometry processing, chemical and medical applications etc. and even in cryptography. Traditionally the hash functions are considered in a form of $h(\mathbf{v}) = f(\mathbf{v})$ **mod** m, where m is considered as a prime number and $f(\mathbf{v})$ is a function over the element $\mathbf{v}$, which is generally of „unlimited“ dimensionality and/or of „unlimited“ range of values.

In this paper a new approach for a hash function construction is presented which offers unique properties for textual and geometric data. Textual data have a limited range of values (the alphabet size) and „unlimited“ dimensionality (the string length), while geometric data have „unlimited“ range of values (usually $(-\infty, \infty)$), but limited dimensionality (usually 2 or 3).

Construction of the hash function differs for textual and geometric data and the proposed hash construction has been verified on non-trivial data sets.

Categories and Subject Descriptors

H.3.1 [Information Storage and Retrieval]: Content Analysis and Indexing; H.3.3 [Information Storage and Retrieval]: Information Search and Retrieval; [Theory of Computation] Miscelaneous

General Terms

Your general terms must be any of the following 16 designated terms: Algorithms, Performance, Experimentation, Verification, Theory.

Keywords

Hash function, textual data, geometrical data, indexing, data retrieval,

1. INTRODUCTION

Hashing techniques are very popular and used in many applications. Their main advantage is a fast retrieval, in the ideal case with $O(1)$ complexity. Unfortunately, in text processing a geometrical data processing is quite different as hash functions can form long buckets that cause unacceptable run-time in real situations. In geometric data processing we need to process $10^6 - 10^8$ of points consisting usually of three, i.e. $< x, y, z>$ coordinates, or more coordinates. The situation is even more complicated by the fact that range of values for each coordinate is generally “unlimited”, i.e. the interval $(-\infty, \infty)$ has to be considered. In the text processing case, the dimensionality is “unlimited”, i.e. a string can be very long, e.g. the titin protein is described by the word *Methionyl-threonylthreonylglutaminylarginyl...isoleucine* which consist of 189,819 characters, but the range of values for each dimension is limited to the size of the actual alphabet.

It can be seen that there are significantly different requirements from those two applications to the hash function construction and hashing method in general.

In the following text we explain how the hash function is constructed in general and then how the hash function is constructed for textual and geometric data. Experiments are described and obtained results are evaluated.

2. HASHING TECHNIQUE

Hashing technique is based on an idea, that there is a hash function $h(\mathbf{v})$ which gives a unique address to a table for the given primitive $\mathbf{v}$ with an acceptable “sparsicity” of the table. This is the idea of perfect hashing [1] or nearly perfect hashing [9], which is not usable for larger data sets. In practical use hash functions [3-4], [6-7] do not return unique addresses for different primitives. It results to buckets, which might be pretty long and lead to unacceptable results in indexing and retrieving items from data sets.

There are actually two problems that should be resolved if the hash function is to be designed, i.e.:

- We should avoid the overflow operation in hash function computation. It is quite severe requirement which is quite hard to implement in general case.
- We should use the whole size of the table and reduce the bucket/cluster lengths as much as possible. The maximal and average cluster length should be as low as possible (cluster is usually implemented as a list of primitives for the cases when the hash function gives the same value).
- The hash function must be as simple as possible in order to have very fast evaluation.

There are several additional requirements that differs from application to application, e.g. distributed hashing etc.

In the following we show the approach taken on geometrical data case first and then how construction of the hash function hould be made for textual data.

3. HASH FUNCTION CONSTRUCTION

3.1 Geometric case

Let us consider the geometrical case first. In this case the dimensionality of the given item is given, usually 2 ($<x,y>$) or 3 ($<x,y,z>$), but the range of values is "unlimited".

Let us consider recommended hash function for geometrical purposes [2].

The original hash function was defined as

$$Index = int\left(3\frac{int(|X| * Q)}{Q} + 5\frac{int(|Y| * Q)}{Q} + 7\frac{int(|Z| * Q)}{Q}\right) \boldsymbol{mod}\, m$$

where: *int* is the conversion to integer - the fraction part of the float is removed, Q defines sensitivity - number of valid decimal digits (numerical error elimination) - for 3 decimal digits set Q = 1000.0, *m* is the size of the hash table that is determined as described later, but generally as 2^k for fast evaluation of the **modulo** and **division** operations, *X*, *Y*, *Z* are co-ordinates of a vertex.

It should be noted that in the graphical data case the number of processed primitives, e.g. points, can easily reach $10^6 – 10^8$.

The hash function shown above uses very simple formula that is recommended in many publications for small or medium data sets. Nevertheless when the property of the hash function was experimentally verified for elimination of duplicities of points, it has not proved good properties for larger data sets, see Tab.1. and Fig.1 - 2. Fig.3 - 4 presents rendered data sets. The experiments proved that the function has relatively stable properties nearly without significant influence of the coefficient Q.

Table 1. Typical characteristic of the original hash function

File	Number of triangles	Original number of vertices	Final number of vertices	Maximal cluster length
CTHead.stl	555 411	1 666 233	278 856	356
Gener.stl	500 000	1 500 000	50 002	577
Teapot.stl	159 600	478 800	80 202	110

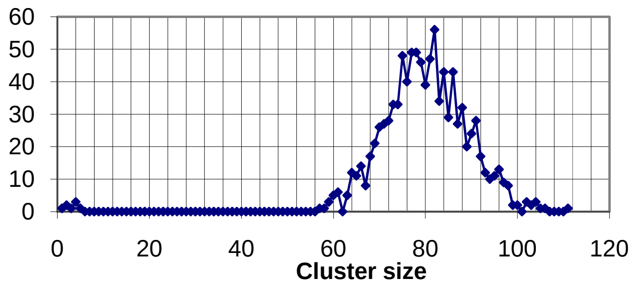

Figure 1. Number of clusters for precision Q=7 decimal points for Teapot data set

One disadvantage of this hash function is that the coefficient Q depends on the data and can lead to mixing some vertices together. The second disadvantage is that the argument of the hash function can easily overflow.

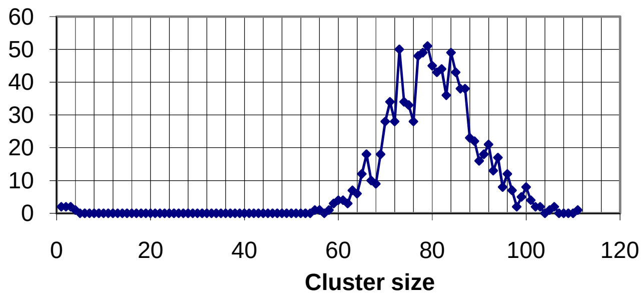

Figure 2. Number of clusters for precision Q=9 decimal points for Teapot data set

Figure 3. Teapot data set

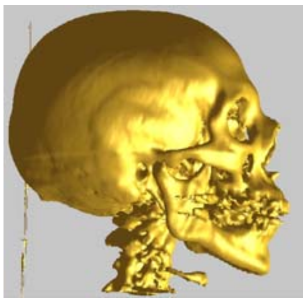

Figure 4. CT Head data set

Data analysis proved that

- it is not reasonable to remove the fraction part from the co-ordinate value as it helps us to distinguish co-ordinates better,
- it is necessary to remove all coefficients that depends on data set somehow – it increases the application stability
- the available memory has to be used as much as possible to get larger hash table,
- the hash function should not be static one - it should be dynamic according to currently available memory, but generally the size of the hash table can be fixed.

Taking into account required properties of the hash function, several functions have been derived in the general form.

$$Index = int\big((\alpha X + \beta Y + \gamma Z) * C\big)\ \boldsymbol{mod}\ m$$

Where: α, β and γ are coefficients of the hash function, e.g. 3, 5 and 7 (not necessarily integer numbers), C coefficient is a scaling coefficient set so that the full range of *integer* values is used, i.e. maximum range of the interval $<0, 2^{32}-1>$ or $<0, 2^{64}-1>$ is used. In the following we consider the interval $<0, 2^{32}-1>$ only.

For a simplicity assume that all co-ordinates *X* are from the $< 0 , X_{max} >$ interval, similarly for others. Then we can compute maximal value ξ that can be obtained from the formula as

$$\xi = \alpha * X_{max} + \beta * Y_{max} + \gamma * Z_{max}$$

Because the overflow operation must be avoided and also we must use the whole size of the table, the C coefficient must be determined as

$$C = \min \{ C_1 , C_2 \}$$

where: $C_1 * \xi <= 2^{32} – 1 \qquad C_2 = 2^{32} - 2^k$

In order to get a maximal flexibility of the hash function we must use the whole address space interval (in our case $<0, 2^{32} – 1>$), influence of C_1 coefficient, and maximum of available memory, influence of C_2 coefficient.

So far we have dealt with the hash function property regardless to the length of the hash table. It must depend on the size of data we are going to process.

It is well known that the length of the table and estimated length of a cluster is in relation with the *load factor* α, see [2] for details. If we consider the *load factor* $\alpha = 0,5$ we can expect cluster length about 2,5.

The length m of the hash table can be expressed as $m \geq N / \alpha$

where: *load factor* - $\alpha = 0,5$ used; the lower value should be used the better spread out, N – number of points.

In practice the value m is chosen as 2^k in order to be able to use the **logical and** operator instead of **modulo** as this solution is much faster.

Nevertheless in some cases the co-ordinate range is not known. In this case the hash function can be easily modified so that co-ordinates are transformed by the function [Pasko95a]:

$$x' = \left(\frac{x}{|x| + k} + 1 \right) * 0,5$$

where: k is a parameter, $0 < k < \infty$.

This function transforms the interval of $x \in (-\infty, \infty)$ to the interval $x' \in (0, 1)$. This transformation preserves the stability of the hash function behavior and it is applied for the y and z co-ordinates as well. The hash function is then constructed similarly as if $\boldsymbol{x_{max}} = \boldsymbol{1}$, now.

The hash function is to be very fast. The formula for ξ is generally for float representation there is no need to convert values to integers.

Experiments proved that if coefficients of the hash function α, β, γ are "irrational", e.g. $(\pi, e, \sqrt{2})$, the length of clusters is significantly shorten. Distribution of cluster length for different data sets is shown at Fig.5 and Fig.6. Fig.7 show that if the table is made longer, no significant changes on distribution of length of clusters happens, see Appendix for more detailed information.

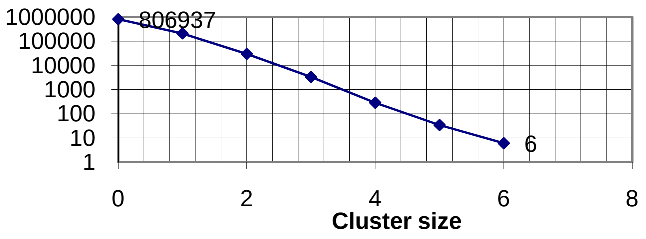

Figure 5. Number of clusters for CT Head $(\alpha, \beta, \gamma) = (\pi, e, \sqrt{2})$

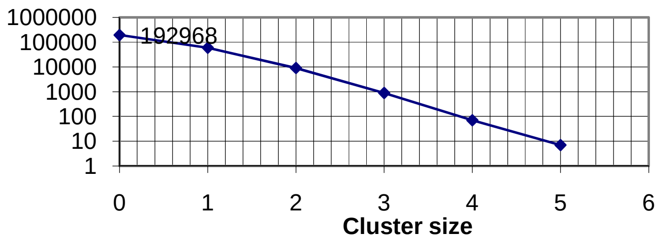

Figure 6. Proposed hash function property for Teapot.stl data

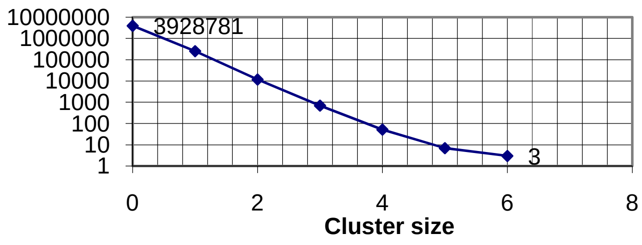

Figure 7. Proposed hash function property for CT Head if the table is longer 4-times

It can be seen that the proposed approach offers better distribution of data, generally shorter clusters and also faster hash function computation. It is independent from a user choice of "magic" parameter Q. Other non-trivial geometrical data set have shown a similar behavior of the hash function.

3.2 Textual data

Textual data are quite different from geometrical data. There are several approaches published in many books. Traditionally the hash functions are considered in a form of $h(\mathbf{v}) = f(\mathbf{v})$ **mod** m, where m is considered as a prime number and $f(\mathbf{v})$ is a function over a string of characters $\mathbf{v}$. It seems to be quite simple, but the ***mod*** operation with a prime is quite a time-consuming.

Let us consider a string of a length L as a $\boldsymbol{L}$-dimensional vector, where each dimension has just one character and therefore a limited set of values given by the alphabet. It should be noted that a string can be very long; the longest word has 189 819 characters!

It means that we have the case with "unlimited" dimensionality, but "limited" set of value at each dimension. The fundamental requirement for any hash function is that a possibility of an overflow in computation has to be avoided in principal. It means that we have to be able to determine a maximal value of the function $f(\mathbf{v})$, i.e. the value before ***mod*** operation is applied.

Considering a simple polynomial function in the form:

$$h(\boldsymbol{x}) = \left(C * \sum_{i=1}^{L} q^i x_i \right) \boldsymbol{mod}\, m$$

We can select a value $q \in (0.1)$ so the sum above is convergent.

The maximal value of $f(\mathbf{x})$ can be easily determined as

$$h_{max} = \left(C * \sum_{i=1}^{L} q^i \varphi_i \right) \boldsymbol{mod}\, m$$

Where φ_i is generally the highest value representing the last character in the given alphabet on the i-th position of the given string (string can consists of different alphabets on each string position). In majority of cases, the φ_i will be the same and representing 'Z' character.

Now we have to determine the constant C so that h_{max} value will be again from the interval $<0, 2^{32}-1>$ or $<0, 2^{64}-1>$.

To be able to compare different hash functions for textual data it is necessary to introduce some general criteria.

Let us assume that there are already *N* items stored in the data structure and *I* is the cluster length. Three basic situations can occur when a new item, i.e. a string, is inserted to the structure:

1. The item is not stored in the data structure and the appropriate cluster is empty. The item is inserted to this cluster. The cost of this operation for all such items can be expressed as
$$Q_1 = 0$$
2. The item is not stored in the data structure so the whole cluster is to be searched and the item is to be inserted to an appropriate cluster. The cost of this operation for all such items can be expressed as (because the cluster of the length *I* must be searched for all items in this cluster *I*-times, value *I* is powered by two)
$$Q_2 = \sum_{I=1}^{I_m} I^2\, C_I$$
3. The item is stored already in the data structure so the corresponding cluster is to be searched and the item is not inserted to the appropriate cluster. Because only half of the cluster is to be searched on average, the cost of this operation for all such items can be expressed as
$$Q_3 = \sum_{I=1}^{I_m} \frac{1}{2} I^2 C_I$$

It is necessary to point out that the cost of the hash function evaluation has not been considered, as it is the same for all cases. The cost of item insertion to a cluster was omitted as the item is inserted to the front of the bucket. The final criterion can be expressed as

$$Q = Q_1 + Q_2 + Q_3 = \frac{3}{2} \sum_{I=1}^{I_m} I^2\, C_I$$

Empty clusters are not considered by this criterion because the hash table length *m* depends on the number of items stored. It can be seen that the criterion *Q* depends on the number of items. We used a relative criterion *Q'* to evaluate properties of hash functions for different data sets with different sizes defined as

$$Q' = \frac{Q}{N}$$

Several experiments with coefficients α, β, γ for the hash function were made recently. These coefficients were taken as decimal numbers and hash function behavior was tested for large data sets with very good results being obtained for geometrical applications. These results encouraged us to apply the hash functions to large dictionaries. In this paper two dictionaries, Czech and English, were used [5]. The proposed approach was also experimentally proved on French, German, Hebrew and Russian dictionaries as well.

There were several significant results from the previous experiments with large geometrical data sets. The most important assumptions of our approach have been:

- large available memory for applications is considered,
- the load factor *f* (will be defined later in this paper) should be smaller than or equal to *0.5,*
- the hash function value should be in the interval of $< 0; 2^{32}-1 >$ before the *modulo* operation is used to get a better spread for all considered items,
- the expected number of items to be stored is in the range $< 10^5; 10^7 >$ or higher,
- hash functions used for strings must be different from hash function used in geometrical applications, because strings can have various length, i.e. non-constant dimensionality, and characters are taken from a discrete value set,
- non-uniform distribution of characters in dictionaries will not be considered in order to obtain reasonable generality and simplicity.

Both hash functions were tested using Czech and English ISPELL dictionaries [5]. The data sets for both languages were taken from the `ispell` package for spell checking. The Czech dictionary contains approx. $2.5 * 10^6$ words and the English dictionary contains approx. $1.3 * 10^5$ words. Several experiments were made including slight modifications for the Czech language.

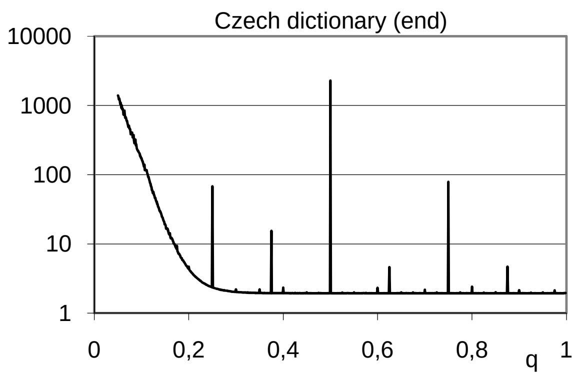

Figure 8. Relative criterion Q' for Czech dictionary

(I_a : min. 1.15, I_m : min. 6)

I_a is an average cluster length, I_m is a maximum cluster length

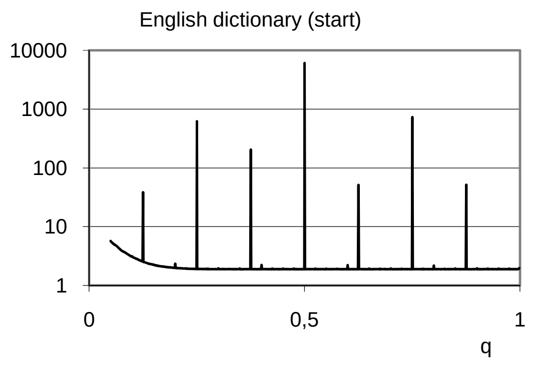

Figure 9. Relative criterion Q' for English dictionary

(I_a : min 1.13, I_m : min 4)

The properties of the hash function are very poor at the interval *(0; 0.3)*. This behavior is caused by the non-uniform frequency of characters in words. The "perfect" overlapping of some words causes the peaks for the multipliers of 0.125. For example: the strings "ab", "aac", "aaae", … will have exactly the same value.

To improve the properties of the hash function and reduce overlapping the coefficient *q* should be an irrational number. However it is not possible to store an irrational number on the computer in the usual way. Thus in the actual implementation the irrational coefficient *q* is approximated using the closest available number on a computer with 64 bit double type. In this instance 17 decimal places were used for the representation of *q*. The experiments were repeated again for "irrational" values of *q* in the interval $q \in (0{,}4; 0{,}9)$. Results of this experiment are presented in Fig.10 - 11.

It can be seen from the graphs that the properties of the hash function were improved and the relative criterion *Q'* has lower value.

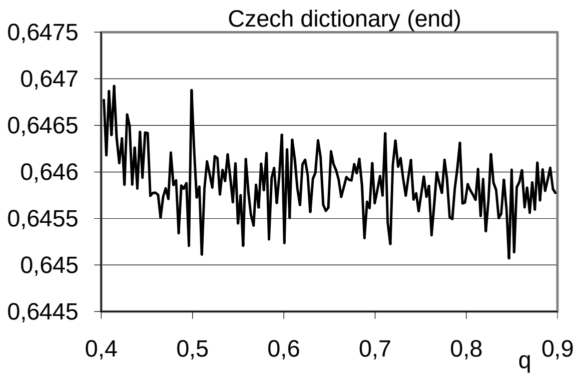

Figure 10. Relative criterion Q' for Czech dictionary

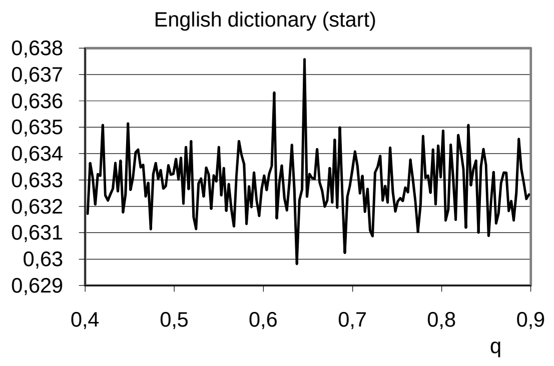

Figure 11. Relative criterion Q' for English

Varying the size of the hash table

Some small improvement can be expected if the size of the hash table is increased and dramatic changes in behavior can be expected if the hash table is shorter than a half of the designed length. Such behavior has been proved in another experiments, see FIG. 6-7; the mark "X" is used for comparison only.

Note that the table sizes were changed accordingly of the hash table varied in interval *<1,048,576; 33,554,432>* for the Czech dictionary and *<65,536; 2,097,152>* for the English dictionary.

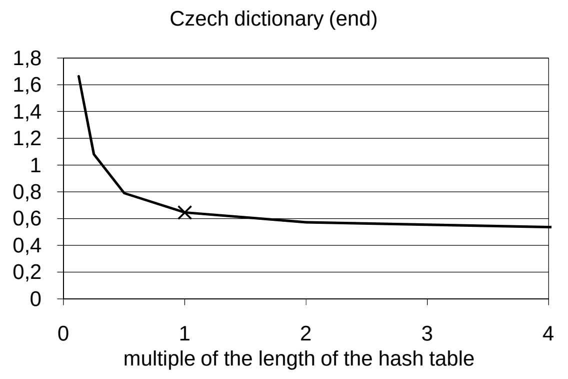

Figure 12. ***Relative criterion Q' for Czech dictionary when varying size of the hash table***
(I_a : 1.02-2.58, I_m : 4-12)

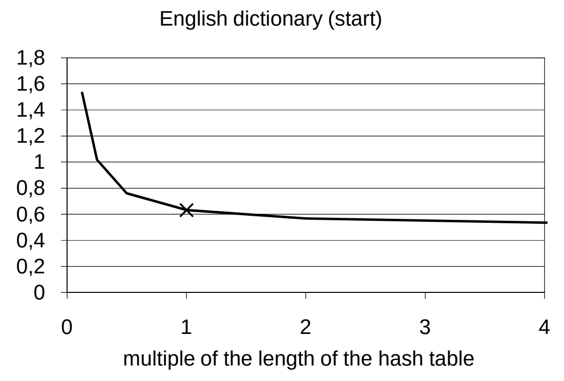

Figure 13. Relative criterion Q' for English dictionary when varying size of the hash table (I_a : 1.02-2.36, I_m : 3-10)

Additional experiments

Conclusions drawn from recent experiments were also supported by additional experiments with other languages. In Tab.3 are the general properties of the dictionaries of the selected languages we used.

Table 2. The general properties of additional languages

Language	*M*	*N*	Table length
French	285 992	220 291	524 288
German	309 838	294 899	1 048 576
Hebrew	10 669	10 669	32 768
Russian	963 212	956 715	2 097 152

M – number of words in the data set, *N* number of unique words

The selected dictionaries were tested using the same hash functions as the Czech and English dictionaries. The Russian dictionary was processed like the Czech dictionary (both of them are Slavonic languages), i.e. the words are processed from the end. The hash function that processed words from the beginning was used for the other dictionaries. The results obtained are presented in Tab. 3.

Table 3. The properties of proposed hash function for additional languages

Language	I_a	I_m	*Q'*
French	1,22	5-7	~0,707
German	1,15	5-8	~0,638
Hebrew	1,16	4-6	~0,652
Russian	1,24	6-8	~0,726

I_a is an average cluster length, I_m is a maximum cluster length

It is obvious from TABLE 3 that the behavior of the proposed hash function with other languages is similar to that with Czech and English dictionaries.

4. CONCLUSION

A new approach to hashing function construction for textual and geometric data was presented with experimental results obtained. The approach was tested on non-trivial data sets – geometrical and textual. Results proved that the hash function described can be effectively used for elimination of duplicities in large textual or geometrical data collections.

The proposed approach can be used in many areas, especially in WEB indexing techniques and time critical textual and geometrical applications.

APPENDIX

Due to use of ***mod*** operation, if the hash table already constructed needs to be shorten, it is is possible to easily modify the hash table for the ½ of its length.

The advantages of the proposed approach are:

- Uses floating point representation that leads to higher stability and robustness is increased significantly
- Simple implementation and fast function evaluation as CPU processors are optimized to FFP operations

Significantly new information is that it irrational numbers for hash function coefficients are used, generally much better cluster length distribution is obtained.

5. ACKNOWLEDGMENTS

The author would like to thank to all who contributed to this work, especially Martin Kuchar and Jan Hradek for experimental verification, to anonymous reviewers for their comments that improved the manuscript significantly. This research was supported by the MSMT CR project LC 06008.

6. REFERENCES

[1] Gettys, T. (2001) : Generating perfect hash function, *Dr. Dobb's Journal,* Vol. 26. No.2, pp.151-155.

[2] Glassner,A. (1994): "Building Vertex Normals from an Unstructured Polygon List", *Graphics Gems*, IV, pp.60 - 73. Academic Press, Inc., Cambridge.

[3] Horowitz,E., Sahni,S.: *Fundamentals of Data Structures*, Pitman Publ.Inc., 1976

[4] Morris,J., Hash Tables: *http://swww.ee.uwa.edu.au/~plsd210/ds/hash_tables.html*

[5] SPELL Dictionaries, *http://ficus-www.cs.ucla.edu/geoff/ispell-dictionaries.html*

[6] Knuth,D.,E. (1969-90) *The Art of Computer Programming, vol. 3, Searching and sorting*, Addison-Wesley.

[7] Korfhage,R.,R., Gibbs,N.E. (1987) : *Principles of Data Structures and Algorithms with Pascal*, Wm.C.Brown Publishers

[8] Mughal,M.S., Nawaz,M., Ahmad,F., Shahzad,S., Bhatti,A.K.,k Mohsin,S: 3D Hash Function for Fast Image Indexing and Retrieval, Computer Graphics, Imaging and Visualization 2007, IEEE 0-7695-2928-3/07

[9] Stein,B.: Principles of Hash-based Text Retrieval, ACM SIGIR 07, pp. 527-534, 2007

[10] Wipke,W.T., Krishnan,S., Ouchi,G.I.: Hash Function for Rapid Storage and Retrieval of Chemical Structures, J.Chem.Inf.Compu.Sci, Voll18., No.1, pp.32-37, 1978

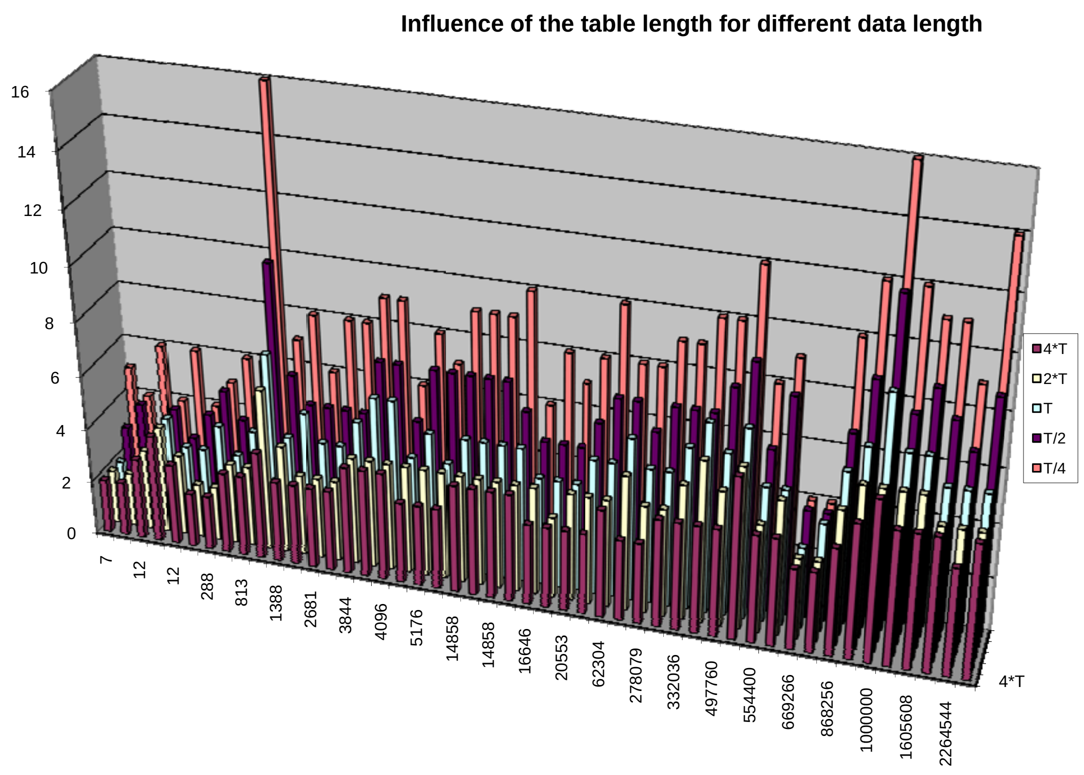

Figure A.1. Maximal bucket length dependence on the number of vertices and the hash table length

Parameters: $(\alpha, \beta, \gamma) = (\pi, e, \sqrt{2})$, **T** is the recommended table length