Maximizing of reusability of test questions by optimizing their randomly generated elements through characteristic text strings

Petr Voborník

***Abstract*—**The article illustrates the unique coding method for describing the properties of different objects of the real and virtual world. This method offers the ability to summarize the basic description of the object in a very short text string comprised only of the basic characters of the alphabet and numbers. These strings describing specific objects can be quickly and easily compared with each other and determine total and partial (differences in specific properties) differences of these objects. This enables us to easily identify the object that is most similar to or most dissimilar from the pattern object. The whole principle of characteristic text strings is subsequently demonstrated on generating random elements in the test questions. A complicated structure of each randomly generated question can be described by the short string thanks to this principle. On the basis of this, such procedures for repeatedly used questions can be suggested that ensure generating of the most different version from versions used previously. The applicability of each of the test questions is thus maximized. Several other areas are proposed in conclusion, where a similar principle can also be applied preferably.

***Keywords*—**Test questions, random elements, Universal Testing Environment, characteristic text strings.

I. Introduction

ONLINE testing system, the *Universal Testing Environment*, allows creating graphically rich, interactive and multimedia enhanced questions of all sorts [1]. In order really not to limit the creative potential of the authors of the test questions, a respective language, QML[1] (based on XML and XAML [2]), was created for defining the structure of the questions. In addition to text, the QML allows the use of vector and bitmap graphics of any kind, animations, and various types of random values at any part of the question. This is achieved by the use of random elements [3] for selecting version of the question, generating random numbers and characters, selecting the text string from multiple variants, and mixing of inner elements structure of the questions.

Like with the selection of questions used in a test [4], it is desirable in this case that the internal mixing is not always entirely random, but it rather provides the opportunity to generate the greatest possible variation of each question that is repeatedly used with the same tested user. Thanks to this, the selection process of random elements in each question should prefer those variations, or their combinations, which would differ the most from the variations used previously.

Determining the degree of difference between individual combinations of random elements should not be computationally complex (e.g. such as multi-distance spatial cluster analysis, see [5]), it should be easily storable in a database, and the structure of such record should be extensible for possible future development of the question.

II. Principle of Comparison of Strings

Firstly, the selected method will be explained using an easily presentable example: comparing of the persons. Table I contains five human properties which will be compared for each person and a description of their data types and ranges (metadata).

Table I – Metadata of persons' properties for comparison

Property	Index	Type	Min	Max	Range	Units	Weight
Height	1	cardinal	40	220	181	cm	1.5
Weight	2	cardinal	1	256	256	kg	0.9
Age	3	cardinal	0	127	128	years	1.0
Sex	4	nominal	1	2	2	-	10.0
Color of eyes	5	nominal	1	6	6	-	2.0

The *range* parameter determines how many possible values the property can take on. The process of its calculation is shown in Equation 1.

$$range_i = max_i - min_i + 1 \quad (1)$$

Properties are differentiated into two main types: *cardinal* and *nominal*. For the cardinal type, differences between values of individual subjects can be directly and precisely calculated, whereas for the nominal values it can only be determined whether the values are identical or not [6]. The calculation of the difference of these two models will therefore vary.

This research has been supported by Specific research project of University of Hradec Kralove, Faculty of Science in 2015.

P. Voborník is with the Department of Informatics, Faculty of Science, University of Hradec Králové, Rokitanského 62, Hradec Králové, 500 03, Czech Republic (e-mail: petr.vobornik@uhk.cz, orcid: 0000-0003-1841-3455).

[1] QML - Questions Markup Language [3]

A. Cardinal values

Each compared subject can essentially be expressed as a vector (e.g. $\vec{a}$ or $\vec{b}$) comprised of the values of individual properties. If each of the cardinal properties had the same weight (e.g. if the difference of one year of age was as significant for the comparison as the difference of 1 cm of height or 1 kg of weight), the difference is in this case formed by a scalar product of a unit vector and a vector composed of the absolute differences of the individual components of the vectors $\vec{a}$ and $\vec{b}$ (see Equation 2).

$$\begin{pmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{pmatrix} \begin{matrix} is \\ different \\ from \end{matrix} \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix} = \begin{pmatrix} |a_1 - b_1| \\ |a_2 - b_2| \\ \vdots \\ |a_n - b_n| \end{pmatrix} \times \begin{pmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{pmatrix} \quad (2)$$

Calculating differences of cardinal properties for entities A and B (Equation 2) can also be expressed using the sum of absolute values of differences of individual properties as shown in the Equation 3.

$$\sum_{i=1}^{n} |a_i - b_i| \quad (3)$$

If differences for each property had a different impact on the overall assessment of the overall similarity of two entities, their weights, which are part of the metadata, can then be taken into account in the calculation ($\vec{v}$, see last column in Table I). Calculation of differences of the cardinal properties of entities A and B with the addition of weights is then shown in Equation 4[2].

$$R_{car.} = \sum_{i=1}^{n} (|a_i - b_i| \cdot v_i) \quad (4)$$

B. Nominal values

For nominal values, although these categories are identified by numbers (for coding), it cannot be determined how much they differ, but only if they are identical or not. If both values are equal, the result is 0, if they are different the result is 1. In this case, either the comparison operator (`if`) can be used for both nominal values, or the *sgn*[3] function to the absolute value of their difference can be applied (see Equation 5). The degree of influence of both types of values on the total difference can be determined by setting the appropriate weight.

$$R_{nom.} = \sum_{i=1}^{m} (sgn|a_i - b_i| \cdot v_i) \quad (5)$$

C. Total difference

The total difference is made up of the sum of the differences of cardinal and nominal values (see Equation 6).

$$R = R_{car.} + R_{nom.} \quad (6)$$

The procedure for particular example of a comparison of two persons is shown in Table II. Differences for individual types of values are calculated with the above equations that are selected according to the types of values (see col. *Type* in Table I).

Table II – Sample of procedure for calculating the difference between persons A and B

Property	Person A	Person B	Difference	Weight	Weighted difference
Height	175	165	10	1.5	15.0
Weight	85	60	25	0.9	22.5
Age	35	25	10	1.0	10.0
Sex	2	1	1	10.0	10.0
Color of eyes	5	1	1	2.0	2.0
Total			**47**		**59.5**

The resulting difference of persons A and B has a value of 59.5. If we considered an extra person C with the same values in all properties as with person B, with the only difference that its height would be 166 cm, then the difference between person A and C would have a value of 58. It follows that the person C would be more similar to person A than to person B. This way, the person that is most similar to or most dissimilar from the pattern[4] person A (from aspect of compared properties) can be found in an arbitrarily large database of people, and the number of comparisons would only be equal to the number of people in the database.

III. Coding

Another requirement to the system of comparison was the ability to easily store values of compared properties in a database. Each value is usually stored atomically in its own field (column of the database table) in a database of persons [7]. In such a database each column must be processes individually, but the necessary calculations can be performed even within the SQL query. For more complex structures, such as the record of internal mixing of the questions, data fields cannot be prepared in advance to effectively cover atomically all the possible combinations. The questions may contain an unlimited number of such combinations. For this reason, only one field was allocated to store combinations of random values.

To store multiple values in a single item, it is necessary to encode them. It is possible to use e.g. the *blob* data type enabling the record of text or binary data of unlimited length,

[2] vector of weights would then replace the unit vector in a vector terms (see Equation 2)

[3] *sgn* function leaves only the input value of 0, all other (positive) values are converted to 1

[4] *pattern* is a term for the entity which other entities are compared with [11]

ISBN: 978-1-61804-322-1

or a limited *varchar*[5]. In such items, text strings in a given coding can be stored. Due to the limitation of the symbols for printed form, only 6 bits of each character of a string would be used to encode values, i.e. the range of 64 (2^6) possible characters. With a binary blob it would then be possible to use the whole range of a byte (2^8 = 256 characters) for individual characters.

Characters for encoding 6-bit values can be chosen arbitrarily, but in general they should be selected from between the 32nd and 126th character of the ASCII table, because these are all viewable in printed form and they are not a subject of the national coding. This eliminates any problems with the encoding of character set, without any further problems when sending it via URL parameter or in a XML file. The string is easily readable by humans and it is also capable of being written in a non-digitized (analog) form.

Selection of characters for the following examples was inspired by the Base64 format [8]. Because the system conversion methods could not be used for this purpose, their order and additional characters were more appropriately adjusted (see Table III).

Table III – The code table for conversion between values (0-63) and characters selected for encoding

0	a	8	i	16	q	24	y	32	A	40	I	48	Q	56	Y
1	b	9	j	17	r	25	z	33	B	41	J	49	R	57	Z
2	c	10	k	18	s	26	0	34	C	42	K	50	S	58	6
3	d	11	l	19	t	27	1	35	D	43	L	51	T	59	7
4	e	12	m	20	u	28	2	36	E	44	M	52	U	60	8
5	f	13	n	21	v	29	3	37	F	45	N	53	V	61	9
6	g	14	o	22	w	30	4	38	G	46	O	54	W	62	-
7	h	15	p	23	x	31	5	39	H	47	P	55	X	63	_

Characters + and / used in the Base64 (see [9]) were replaced by others, because these are subject of coding when it is transferred as the URL parameter. The proposed arrangement of characters is partly legible for humans, because lowercase letters and numbers from 0 to 5 form the first half of the table and the uppercase letters, remaining numbers, and other characters are located in the second half. With a good knowledge of the alphabet, a more precise value can be estimated intuitively out of each character.

Another difference from Base64 is the overall approach towards coding. Base64 losslessly encodes arbitrarily large value to the required number of characters (see the example in Table IV).

Table IV – Process of character encoding from plain "yes" to the Base64 "eWVz"

Data (text)	y								e								s							
ASCII code	121								101								115							
Bit pattern	0	1	1	1	1	0	0	1	0	1	1	0	0	1	0	1	0	1	1	1	0	0	1	1
6-bit number	30				22				21				51			
Base64	e				W				V				z			

In contrast, the proposed method is based on the premise, that the value of each property will be stored as just a one character, i.e. the encoded value will be transferred to the range of 0-63. Larger values will be stored in a lossy format and smaller ones will not use the full range of the character, but thanks to this approach individual properties can be directly individually compared.

For conversion of property values (a_i) to the range of 0-63, each value is first normalized to the range of $\langle 0, 1\rangle$ (n_i, see Equation 7) and then it is multiplied by the maximum value of this range (63). The result rounded to the nearest integer (in this step there is a loss of accuracy) is converted to a character according to Table III.

$$n_i = \frac{a_i - min_i}{range_i - 1} = \frac{a_i - min_i}{max_i - min_i} \tag{7}$$

This procedure of coding can be applied to all numerical values, including nominal, provided the individual categories are first converted to numbers (indexes). This step is necessary for values bigger than 64, while it is not for discrete values of a lower range. With them, the minimum is sufficient to be subtracted from them and the result can be directly used for encoding without using the entire range. This uniform procedure was selected for the following example and for all types of values. Table V shows the process of encoding values of properties of two persons from the previous example.

Table V – Process of encoding values of properties to characters

Property	Person A				Person B			
	Value	n_i	0–63	Char	Value	n_i	0–63	Char
Height	175	0.750	47	P	165	0.694	44	M
Weight	85	0.329	21	v	60	0.231	15	p
Age	35	0.276	17	r	25	0.197	12	m
Sex	2	1.000	63	_	1	0.000	0	a
Color of eyes	5	0.800	50	S	1	0.000	0	a

As shown in Table V, the characteristics of person A are encoded in a "characteristic" text string of "Pvr_S", and person B's to the string of "Mpmaa". From these strings it is apparent at first glance that these are two different people that do not match in any parameter, including the last two nominal values.

[5] *varchar* – database type for a string of a variable length limited from above [12]

IV. Comparison of Characteristic Strings

Comparing of two characteristic text strings can be done in several ways, or in various stages of coding back to the original value. Various weights must be used to achieve the same result in each of these stages.

The easiest way is the absolute differences of values converted directly from the individual characters (see col. "Code values – difference" in Table VI).

Table VI – The process of decoding individual characters of a text string

Property	Characters		Code values (0–63)		
	A	B	A	B	difference
Height	P	M	47	44	3
Weight	v	p	21	15	6
Age	r	m	17	12	5
Sex	_	a	63	0	1
Color of eyes	S	a	50	0	1
Total					**16**

The second option are the absolute differences of values normalized to the range of ⟨0, 1⟩ n_i, i.e. the previous value divided by 63 (see col. "Normalized values – difference" in Table VII).

Table VII – Back-calculation of the original values (a_i) of individual properties (continued from Table VI)

Property	Normalized values ⟨0, 1⟩			Original value (a_i)		
	A	B	difference	A	B	difference
Height	0.746	0.698	0.048	174.3	165.7	8.6
Weight	0.333	0.238	0.095	86.0	61.7	24.3
Age	0.270	0.190	0.079	34.3	24.2	10.1
Sex	1.000	0.000	1.000	2.0	1.0	1.0
Color of eyes	0.794	0.000	1.000	5.0	1.0	1.0
Total			**2.222**			**44.9**

The third option is to compare the original decoded values (a_i) by the means of inverse operation of calculating n_i (Equation 7) as shown in Equation 8.

$$a_i = n_i \cdot (range_i - 1) + min_i = n_i \cdot (max_i - min_i) + min_i \quad (8)$$

Original weights (see Table I) can be used only in the third stage, i.e. when comparing the original values. However, a one-off conversion of weights saves a need of repeated conversion of coded values to the original ones for each property of each compared subject.

V. Question Mix

Initial setting of encoding conditions (metadata) for the specific cases would be performed by someone who would assess ranges and weights best suited for the particular purpose, or would make experimental measurements and subsequently edit the ranges accordingly. For test questions containing various random elements in various positions of the question, the settings would be very difficult for the creator of questions, in some cases (e.g. for mixing of content position) even impossible. The whole system should thus work completely automatedly in the background.

Since the sequence of the same random elements in a question may differ on each occurrence, and some portions may even be completely omitted, fixed identifiers (`id`) were used to detect the elements. On the first processing of a question the metadata (or the data necessary for coding of values) are listed under their identifiers, i.e. types of values and their minimum and maximum[6]. Numeric identifiers[7] of positions (`index`) and values (`val`) are also assigned to these elements (or their items) if necessary. Through them they will then be encoded into the characteristic string. The order of items (individual characters) in the encoded string then determines the order of their record in this list. List of metadata (e.g. see Code 1) is stored in a database directly within the record of the question and, can be expanded if necessary (when adding new elements or uncovering previously inaccessible elements due to random selection).

```
<metadata>
  <versions index="0">
    <ver id="x" val="0" />
    <ver id="y" val="1" />
  </versions>
  <rnd index="1" id="a" type="car" min="10" max="99" />
  <rnd index="2" id="b" type="car" min="1" max="9" />
  <rnd index="5" id="var" type="nom" /> <!– char A-Z -->
  <mix id="m1">
    <itm index="3" id="i1" />
    <itm index="4" id="i3" />
    <itm index="6" id="i2" />
  </mix>
</metadata>
```

Code 1 – Sample of possible record of question metadata to XML

The string containing random values for each use of each question is stored in the database during the generation of the test, along with other statistics for each instance of the question, in a form of a plain text string (*varchar*). An example of the coding of a question values with random elements described by metadata in Code 1 into the string of "`bMyb#ha`" is shown in Table VIII.

[6] this information is directly specified for random values `<rnd>` of the range and in other cases they are not required

[7] identifiers specified by creator of questions may not be numeric

ISBN: 978-1-61804-322-1

Table VIII – Sample of encoding of question random values into the string "bMyb#ha"

Element	Value	Coding process	Character	Index
ver	y	val["y"] = 1	b	0
a	72	$\frac{72-10}{99-10} \doteq 0.7 \rightarrow 0.7 \cdot 63 \doteq 44$	M	1
b	4	$\frac{4-1}{9-1} = 0.375 \rightarrow 0.375 \cdot 63 \doteq 24$	y	2
var	H	index H in {A–Z} = 7	h	5
m1	i2,i1	i1 = 1, i2 = 0, i3 = null	ba#	3,6,4

If any of the values in the list of a current question mix was omitted due to the random selection, it would still have to be included in the code string, so that its individual positions would always correspond to the same elements. For these cases, a special character # (hash, or sharp) will be used as the equivalent of the *null*[8] value, in the following example. It indicates that the item's value is not represented in the string[9]. In this case there are three basic approaches for comparing this value with others.

- Mark both strings as incomparable.
- Set a zero difference for the given property in the comparison of these specific characters.
- Evaluate the fact that the property exists in one case and not in the other, as the maximal difference.

The choice of the optimal approach depends on the particular application. The third option (maximum difference) was selected for comparing question mix.

A. Multiple comparisons

The process of comparing two entities, or their characteristic strings, was explained in the previous chapter. This method can be applied to an arbitrarily extensive list and thus a subject can be found that is most similar or dissimilar to the pattern object. The aim of comparing a question-mix is to determine the difference not only between two subjects but between one entity (generated question) and the group of entities (previously used questions).

If the same tested user has been previously asked the same question more times, then only the last five cases[10] are used for comparing, as it helps to save computational cost. The pattern string is compared with each of them and the total difference is the sum of these values. Individual comparisons, however, do not have the same weight. A similar principle as was the forgetting component at the selection of test questions (see [4]) is used in this case. In this case the weight is only repeatedly lowered[11] by one fifth[12] in order to accelerate the calculations.

B. Most different random values

There are more ways to create the most diverse question to its earlier versions. For example, the whole question can be generated several times and the version which is most different from all the previous versions will be used. This will be evaluated by the comparing of their characteristic strings. Respecting individual values during question creation can also be used, which is possible due to their division into individual characters. This method was selected, because of the complexity of repeated parsing of QML. Therefore, since there is no comparison of a whole question but only at the level of individual elements, weights and mutual ranges of these parameters do not have to be taken into account in the calculation.

C. Versions

Random variant of a question based on the versions <versions> defines a finite number of possibilities, one of which is randomly selected. Individual elements of the question are then shown or hidden, accordingly to which version was randomly selected. In terms of the type of values, versions are *nominal*.

For the nominal values such encoding is used which does not convert the value to the range of 0-63 (unlike the previous example of people comparison), but values are indexed by integers from 0 to the count of versions. The impossibility to include all versions into the calculations if their enumeration contains more than 64 variants is undoubtedly the disadvantageous aspect, but this is only a theoretical possibility. The advantage is the possibility to extend the list of other items at any time (up to 64), without invalidating the data from previously completed testing.

Selection of the version is realised by the means of five random choices from all possible versions. If any of them was not used in the previous five representations of the question, this version is declared as the best and the process is done. However, if all five randomly selected "candidates" were used in the previous five cases, the best of them is determined by the highest total sum of weighted nominal differences from the previous selections (see example in Table IX).

[8] *null* is a special value applicable in databases and some object-oriented programming languages across the data types which indicates that the field of the database record or variable is not set to any value [7]

[9] when encoding to the range of 256, this range can be reduced by one (255) and the last value reserved for the *null*

[10] any number of previous cases can be used for comparison, or even all of them

[11] the time component described in the mentioned source can be also used

[12] the latest variant of the question has a weight of 1, 0.8 for the penultimate, 0.6 for the preceding, etc. and the fifth (the oldest in the selection) has a weight of 0.2

ISBN: 978-1-61804-322-1

Table IX – Sample of selection of the most diverse random version (out of five) in relation to the five previous selections

Question		Prior versions		Random versions					
Date of use	Weight	Character	Version	1	4	5	3	1	Weighted nominal difference
2015-02-08	1.0	b	1	0.0	1.0	1.0	1.0	0.0	
2015-02-01	0.8	f	5	0.8	0.8	0.0	0.8	0.8	
2015-01-26	0.6	e	4	0.6	0.0	0.6	0.6	0.6	
2015-01-05	0.4	d	3	0.4	0.4	0.4	0.0	0.4	
2014-12-15	0.2	f	5	0.2	0.2	0.0	0.2	0.2	
Total				2.0	2.4	2.0	2.6	2.0	

The example in Table IX shows the procedure of selecting random version out of five possibilities. All proposed versions (2x1, 3, 4 and 5) were used in the previous five cases. First, the previous uses of questions are sorted in the descending order by the date, and according to this order, weights are assigned to them. In the next step, for each of these selections the relevant character is decoded from the question string to the specific numeric identifier of particular used versions. Subsequently, nominal differences between each previous and proposed versions are calculated (0 for identical, 1 for different) and these results are multiplied by weights reflecting topicality of previous choice. Their sum for each "candidates" is the decisive factor for determining the "winning" version. That is the fourth in this example with the ID number of 3 (`val="3"`), because its overall difference from the previous five versions has a highest value of 2.6.

D. Random values

Generating random values `<rnd>` for the question is also repeated five times. All random values from these five (r_j) are compared (the absolute value of the difference) with values of five previous versions (p_i), and these differences are multiplied by weights (v_i) reflecting the 'age' of questions (see Equation 9).

$$d_{ij} = |r_j - p_i| \cdot v_i \tag{9}$$

The random value (r_j) with the highest total sum of weighted differences is included into the question ($\sum_{i=1}^{5} d_{ij}$, see example in Table X).

Table X – Sample of selection of the most different random integer value (from 1 to 100) with respect to the five previous values

Question		Prior values		Random values					
Date of use	Weight	Character	Value	5	32	17	98	61	Weighted difference
2015-02-08	1.0	s	29.29	24.3	2.7	12.3	68.7	31.7	
2015-02-01	0.8	Y	89.00	67.2	45.6	57.6	7.2	22.4	
2015-01-26	0.6	1	43.43	23.1	6.9	15.9	32.7	10.5	
2015-01-05	0.4	d	5.71	0.3	10.5	4.5	36.9	22.1	
2014-12-15	0.2	p	24.57	3.9	1.5	1.5	14.7	7.3	
Total				118.7	67.2	91.8	160.3	94.1	

The example in Table X shows the procedure of a particular case for selection of the random value in the range from 1 to 100, beginning with the decoding characters to the previous values and ending up with determining the best values from the five possibilities. In this example, this is the fourth ($j=4$) possibility (98), its total difference from the previous five selections is the highest (160.3).

Random selections from the enumerations are treated as nominal, e.g. for values of type *string* (text) or *char* (character). Their selection is similar to the choice of the optimal version. Even in this case it is not necessary to repeat the selection five times, because if the selected value was not in the previous five selections, it can be used directly and the process of selection for this item is over.

E. Random selection

The procedure of mixing the content of the question `<mix>` is more complicated [10]. Each item from a mix `<item>` selection has its own identifier under which (with the prefix of an identifier of the mix or as its sub-element as in Code 1) it is registered in question metadata. A value that is stored below is the position of the item in this mix[13]. If any item is omitted (due to the numerical limitations for the selection), a *null* value (#) is inserted on the position of the item in the string. When an assigned position is compared to a *null* value, the difference is the total count of selected items for the current mix element (becoming a maximum mismatch). When two valid values are compared, their relationship is evaluated as the absolute value of their difference.

Table XI – Example of the procedure of determining the most diverse variant of random selections (4 of 6 items) relatively to the previous five

Prior testing			Random selection																					
Weight	String	Items positions	V1							V2							V3							
		1 2 3 4 5 6	1	2	-	3	-	0	vΣ	3	-	-	2	0	1	vΣ	0	2	-	-	1	3	vΣ
1.0	#c#bda	- 2 - 1 3 0	4	0	0	2	4	0	10.0	4	4	0	1	3	1	13.0	4	0	0	4	2	3	13.0
0.8	bcda##	1 2 3 0 - -	0	0	4	3	0	4	8.8	2	4	4	2	4	4	16.0	1	0	4	4	4	4	13.6
0.6	d#acb#	3 - 0 2 1 -	2	4	4	1	4	4	11.4	0	0	4	0	1	4	5.4	3	4	4	4	0	4	11.4
0.4	#cb#ad	- 2 1 - 0 3	4	0	4	4	4	3	7.6	4	4	4	4	0	2	7.2	4	0	4	0	1	0	3.6
0.2	#bcd#a	- 1 2 3 - 0	4	1	4	0	0	0	1.8	4	4	4	1	4	1	3.6	4	1	4	4	4	3	4.0
Total									39.6							45.2							45.6

Random selection																							
V4								V5															
1	2	3	-	0	-	vΣ	3	2	1	-	0	-	vΣ										
4	0	4	4	3	4	19.0	4	0	4	4	3	4	19.0										
0	0	0	4	4	0	6.4	2	0	2	4	4	0	9.6										
2	4	3	4	1	0	8.4	0	4	1	4	1	0	6.0										
4	0	2	0	0	4	4.0	4	0	0	0	0	4	3.2										
4	1	1	4	4	4	3.6	4	1	1	4	4	4	3.6										
						41.4							41.4										

[13] also in the case of coding positions of items of random selection is not resized to the range of 0-63 for the possibility of future extensibility of the selection for other items

ISBN: 978-1-61804-322-1

Even in this case, the five possible variants of the selection with different positions of the individual items are generated, and these are compared with the previous five. The differences of individual items within mixes are summed and multiplied by the weight reflecting the topicality of previous version (same as in the previous cases). Their sum for each "candidate" of random selection is then the decisive factor for determining the "winning" variant, which is again the highest value of this sum (see Table XI).

The example in Table XI shows the procedure of a particular case of random selection, beginning with the decoding of previous selections from the fragments of the characteristic strings and resulting in determining the best variant of selection out of five. This is V3, its total difference from the previous five selections is the highest with the value of 45.6. According to this variant, out of the 6 items, the 1st, 5th, 2nd and 6th item (in this order) should be included in the selection.

For better understanding of the relationship between the values in Table XI, three related values are framed. These are the 4th item from V2 selection (2, i.e., the 3rd in order will be registered under the index 4), the same item at the penultimate version of the question (0, i.e., there was a 4th item in the 1st place) as well as the absolute difference in the intersection of the two coordinates (|2-0| = 2, i.e., the proposed position of the item in comparison to the penultimate question is shifted by 2 positions).

VI. Conclusion

The characteristic text strings allow us to encode values of properties of different objects into legible characters and to mutually compare their similarities. The whole process was demonstrated on comparing the basic characteristics of two different people. The procedures and ways of practical use of such strings were also presented used to generate more diverse, rather than purely random test questions for the same tested user. The same procedure can also be used for various tested users during mass preparation of questions at the same time, for a single IP address or a computer lab, as a precaution against possible cheating. Other possible uses were outlined, including various ways of implementation of the entire process or its individual parts.

Thanks to the code strings, database structures can be simplified. Support for searching and comparing these strings directly by SQL functions should not be difficult to implement in some database systems.

Comparing, however, is not always performed on the large database of different subjects, but thanks to the ease of portability of characteristic strings can be realised individually "in the field", e.g. using a mobile phone. In addition to text characters for storing values, it is also possible to use other data structures, e.g. barcode or QR code.

Thanks to the separation of the individual properties to the individual characters, it is also possible to compare only certain parts of them, independently of the rest without decoding the whole string.

Areas of application are broad, and this principle can be used wherever it is necessary to store and compare the cardinal or nominal characteristics. These may include e.g. parameters of goods, searching for people, vehicles, dating etc. Strings encoding characteristics of objects with possibility of fast mutual comparisons are also a very important part of genetic algorithms, where this approach could also be applied.

Acknowledgment

Author thanks to Josef Lejp for correction of his English translation.

References

[1] P. Voborník, "Universal Testing Environment as an External Tool of Moodle," in *10th International Scientific Conference on Distance Learning in Applied Informatics (DiVAI 2014)*, Štúrovo, Slovakia: Wolters Kluwer, 2014, pp. 215-225, ISBN 978-80-7478-497-2. Available: http://download.petrvobornik.cz/docs/clanky/2014-divai.pdf

[2] M. MacDonald, *Pro Silverlight 5 in C#*, 4th ed., New York: Apress, 2012, ISBN 978-1-4302-3479-1.

[3] P. Voborník, "Universal Testing Environment," Ph.D. dissertation, Faculty of Informatics and Management, University of Hradec Králové, Hradec Králové, Czech Republic, 2012. Available: http://download.petrvobornik.cz/docs/disertace.pdf

[4] P. Voborník, "The computing model for intelligent selection of the testing questions," in *The proceedings of the 14th international conference MEKON 2012*. Ostrava: VŠB-TUO, 2012, ISBN 978-80-248-2552-6. Available: http://download.petrvobornik.cz/docs/clanky/2012-mekon.pdf

[5] M. Meloun and J. Militký, "Přednosti analýzy shluků ve vícerozměrné statistické analýze," in *Sborník přednášek z konference Zajištění kvality analytických výsledků*, Medlov: 2 THETA, 2004, pp. 29-46, ISBN 80-86380-22-X.

[6] F. Pavelka and P. Klímek, *Aplikovaná statistika*, Zlín: FaME, 2000, ISBN 80-214-1545-2.

[7] P. O'Neil, *Database: Principles Programming Performance*, San Francisco: Morgan Kaufmann, 2014, ISBN 9781483184043.

[8] L. L. Wen, X. Z. Ruo, K. Jian, T. Ling and H. C. Guang, "A Design of Improved Base64 Encoding Algorithm Based on FPGA," in *Applied Mechanics and Materials*, vol. 513-517, pp. 2220-2223, Feb. 2014, ISSN 1662-7482.

[9] D. Ashley. (2010, Sep 20). Obfuscation used by an HTTP Bot. Information Security Office [Online]. Available: http://security.utexas.edu/consensus/Obfuscation.pdf.

[10] Š. Hubálovský and J. Šedivý, "Algorithm Development and Computer Simulation of Position Order Decoding of Mastermind Board Game," in *Applied Mechanics and Materials*, vol. 333-335, pp. 1353-1356, Jul. 2013, ISSN 1662-7482.

[11] V. Strnadová, *Interpersonální komunikace*, Hradec Králové: Gaudeamus, 2011, ISBN 978-80-7435-157-0.

[12] P. Císař, *InterBase / Firebird - Tvorba, administrace a programování databází*, Brno: Computer Press, 2003, ISBN 80-7226-956-1.

Petr Voborník was born in the Czech Republic in 1982. He received a master degree (Ing.) in Information Management and a doctoral degree (Ph.D.) in Information and Knowledge Management from the University of Hradec Králové in 2006 and 2012. He is now an Assistant Professor at the Department of Informatics, Faculty of Science, University of Hradec Králové. His current research interests include developing new algorithms and mini-languages for optimization of electronic testing for his Universal Testing Environment. He also participates in several researches as a programmer, he teaches and popularizes programming and he creates independent applications and games.