Automatic Alphabet Recognition

Maayan Geffet and Yair Wiseman and Dror Feitelson
School of Computer Science and Engineering, Hebrew University
Jerusalem, Israel
{mary, wiseman, feit}@cs.huji.ac.il

Abstract

English texts are usually written in the ASCII standard. Unlike the English language, many other languages which have other characters sets, don't have one standard. This plurality of standards causes problems in various information retrieval tasks, especially in a web environment, where one may download a document in an unknown standard. This paper suggests a purely automatic way of finding the standard which was used by the document writer, based on the statistical letters distribution in the language. The algorithm was applied on various types of corpora in Hebrew, Russian and English and provides a full solution to the stated problem in most cases.

Keywords: Alphabet recognition, information retrieval, national languages, statistical letters distribution, vector-space model, positions vectors, environmental vectors.

1 Introduction

As the Internet has become a part of our life, people all over the world access the web and look for information on a daily basis. The importance of national languages support is growing together with the growing popularity and spread of the network.

The ISO-8859 standard defines the mapping of the higher 128 codes (as ASCII only defines 7-bit codes) for Latin alphabets and for many other languages, including Cyrillic, Arabic, Greek, and Hebrew alphabets (these use different numbers: 8859-5 is Cyrillic, 8859-6 is Arabic, etc.) [7]. An alternative is the Unicode standard, which uses 16-bit codes to provide unique codes to the symbols needed for all commonly used alphabets in the world.

Regrettably, these standards have not come to dominate usage, as opposed to the dominance of ASCII for English. For example, the standard used by Microsoft software for Hebrew characters, known as Windows-1255 [11], does not conform to ISO-8859. Thus it is relatively common to download a document in a non-English language, and find that it uses an unexpected encoding of letters. The result is that the wrong glyphs are displayed, and the document cannot be read (fig. 1.1). Such cases occur quite often while exploring non-English sites on the web. The typical solution is to try different fonts supported by the browser, with the hope that one of them uses the same mapping as the document.

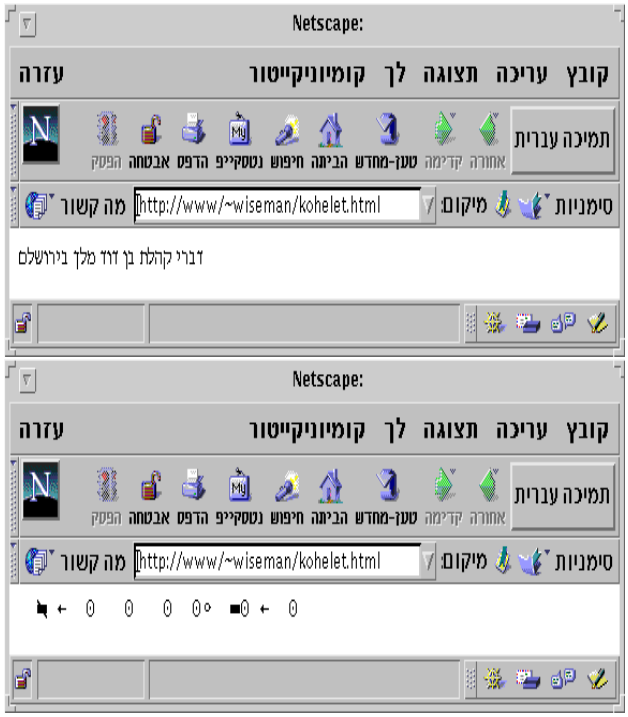

Figure 1.1: Results of displaying a Hebrew document with different fonts, using a Hebrew version of Netscape [12]. The top one provides the correct decoding of the letters. The second font does not define any glyphs for the higher 128 codes, so the browser displays some special graphical symbols.

Another typical problem for Semitic languages is Bi-directional text, where some text is right-to-left (e.g. words) and some other (e.g. numbers) is left-to-right. The visual representation stores the text as it should be displayed on the screen device. The existing standards for web pages, HTML-4 [4] and XHTML 1.0 [10] (latest version, which is a reformulation of HTML 4 in XML

1.0), support Bi-directional text as defined in Unicode's Bi-directional algorithm [11], [8]. According to this standard the characters are stored in the order they are typed by the human user (logical representation) and users are supposed to supply a *language*, a *charset* (to identify the encoding standard) and a *dir* (the direction of a page - '*rtl*' or '*ltr*') parameters in their HTML pages in order to allow a browser to process a document correctly.

Unfortunately, this simple solution does not work in practice since most users don't provide this information. Also, many Hebrew documents are encoded visually in various non-standard ways, and therefore cannot be displayed even by standard-conforming browsers.

Our goal in this paper is to develop a methodology that can be used by a browser to automatically determine which encoding was used in a document. This ability of the system is critical for the successful information retrieval in the multi-lingual environment.

2 Template-Based Recognition Approach

The paper's aim is to find an efficient statistical solution to the alphabet recognition problem, with no language-dependent knowledge, such as morphological and syntactic rules or dictionary usage. The basic idea is to precompute universal templates of letters distribution in a language based on various corpora examples, and then compare them to the statistics of a given document that needs to be decoded.

2.1 Positions Vectors

It was shown in [19] that the distribution of letters in a given human language is generally similar in many texts.

Unfortunately, a good recognition cannot be achieved by this information. There are groups of 3-4 letters (e.g. A, B, $, and N) which have very close frequencies, so it will be hard or even impossible to distinguish between them. We are interested in getting more detailed characteristics of each letter in the language. Our first attempt was to construct a "*positions vector*" for each letter, by counting its occurrences at every position in the word separately.

More formally, define $count_{l,i}$ to be the number of times letter l appears in position i in the word. We distinguish two cases : when position i is the last one in the word, and when it is not; thus $count_{l,i} = count_{l,i}^{not-last} + count_{l,i}^{last}$. The position vector P_l is then defined to be $(a_1, a_2, ..., a_{19}, a_{last})$, where $a_i = \frac{count_{l,i}^{not-last}}{total} * 100$, and $a_{last} = \frac{\sum_i count_{l,i}^{last}}{total} * 100$. $total$ denotes the total number of all the letters in the text and words are assumed to be shorter than 20 letters long. Based on this information, the position of each letter can be discovered.

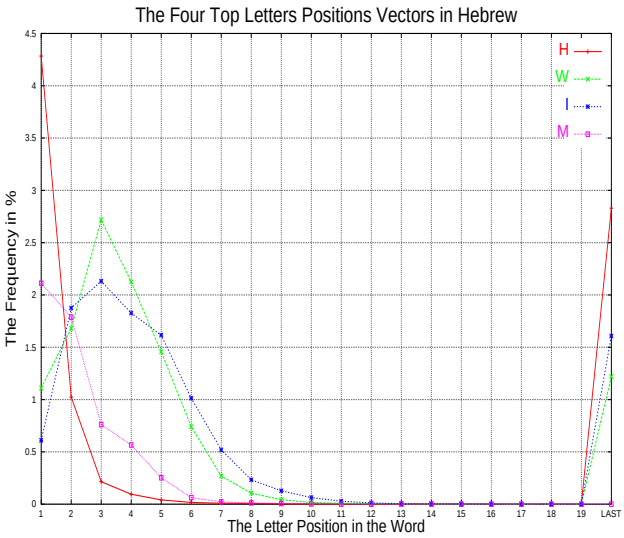

Figure 2.1: ***Positions vectors*** for the four most frequent letters (I, W, H, M) in the Hebrew Scientific Journals (HSJ in Table 1). While in this and other graphs the x axis is actually discrete, plotting lines that connect the values aids the eye in making the comparison. As we can see the vectors are very different.

Comparing the vectors of frequencies of the letters, rather than single values, provides much more accurate results (fig. 2.1, 3.2).

2.2 Environmental Vectors

So far we looked at the text as a 0th order Markov Chain, as we treated each letter independently. However, it may be helpful to consider the closest neighbors of the letter in order to identify it and for that purpose to extend our model to a higher order Markov Chain.

Markov Chains are commonly used in a human language processing to define compression rules [1], [3] and to compute the probability of the next letter by its precedents [6], [20]. For example, in English "u" tends to appear after "q", while "x" almost never occurs after "z" [19], [21]. Such rules are a very strong feature in English, but less so in Hebrew writing, which puts almost no restrictions on letters combinations, since it does not contain vowels. Nevertheless, the differences in the probabilities of different pairs are sufficient to aid in recognition.

First, we collect information about all possible pairs of letters occurrences in the corpus. This data may be viewed as a matrix M of the size: $\mid Alphabet \mid \times \mid Alphabet \mid$, where every cell $M_{i,j}$ contains the frequency of the corresponding pair of letters. The next step is to use this matrix of pairs to find characteristics for individual letters. We notice that rows and columns of M represent the successors and precedents vectors of all the letters, respectively. So here again we took a vector-space-model

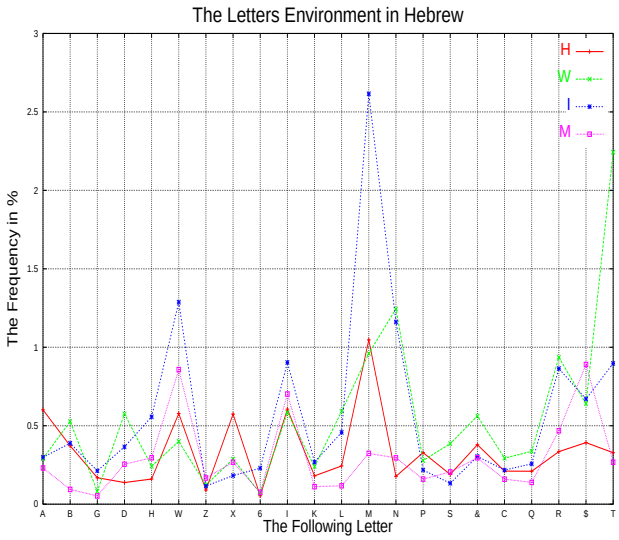

Figure 2.2: ***Environmental vectors*** of the most frequent letters in the Hebrew Scientific Journals (HSJ in Table 1). The difference between the letters is even bigger than as determined by the ***positions vectors*** in fig. 2.1.

to represent a letter's closest environment (fig. 2.2, 3.3). For each letter l in the alphabet, we define its *"environmental vector"* to be its row M_l of frequencies of different successors: $M_l = (a_{l,A}, a_{l,B}, ..., a_{l,|Alphabet|})$, where $a_{l,s} = \frac{count_{l,s}}{total} * 100$, where s is l's successor in the text.

The main problem of both algorithms is ambiguity, when several distinct letters are mapped to a single letter in the other text.

Note, that we still did not use the information contained in the columns of the matrix, the precedents *"environmental vectors"* (PM). This redundant information is useful to disambiguate the results, thus increasing our model to 2nd order Markov chains.

2.3 On-line Matching

Once the off-line construction of templates for various languages is completed, the system can start to work on-line, getting new documents and matching their vectors to the templates.

The matching procedure receives two sets of vectors, V_1, V_2 generated from two texts, and output is a set of pairs, that are the closest to each other: $PS = \{(v, w) : v \in V_1, w \in V_2, f(v, w) = min_{w\prime \in V_2} f(v, w\prime)\}$. V_1 usually stands for the template vectors and V_2 for the new text vectors set.

Another question to be discussed in this context is the f function, i.e. the vectors distance metrics. We experimented with two versions of the norm formula: $f_1 = \|V_1 - V_2\|_1 = \sum_{1 \le i \le n} \mid x_i - y_i \mid$ vs. $f_2 = \|V_1 - V_2\|_2 = \sqrt{\sum_{1 \le i \le n} (x_i - y_i)^2}$. The latter norm decreased the accuracy by up to 10% as demonstrated in fig. 3.4, so we chose f_1 as the better metric for our purposes.

Mapping is executed in both directions: $V_1 \Rightarrow V_2$ and $V_2 \Rightarrow V_1$ to reduce ambiguity and make sure that every letter gets a pair from the other set. The two results are then merged. This helped increase the hit ratio by up to 6% in 25% of the cases as shown in fig. 3.4.

2.4 Combined Method - The Final Version

Another way to eliminate ambiguity is to *combine* the two proposed algorithms, the *"environmental vectors"* and the *"positions vectors"*. The *"combined"* method resulted in better accuracy percentage, as shown in Table 1. The final version of the proposed algorithm is summarized below.

1. Compute positions (P_1) and environmental (M_1, PM_1) vectors for the templates set. This is done off-line.
2. Get a new document from the user.
3. Compute positions (P_2) and environmental (M_2, PM_2) vectors for the document.
4. Pick a template, either the default "Newspapers Style" or according to a user selection.
5. Compare the successors environmental vectors of the template, M_1, to those of the document, M_2, using f_1.
6. For letters that got none or several mappings do:
   - Execute the 4th step mapping in the opposite direction:
     $M_2 \Rightarrow M_1$
   - Check:
     (a) If a letter l_j^2 in M_2 was mapped to n different letters in M_1 and if it got a unique mapping l_i^1 now (which is one of those n):
       – Add a pair (l_i^1, l_j^2) as a match in PS (the final matched pairs set).
     (b) If a letter l_j^2 in M_2 was not mapped at all and if it got a unique mapping l_i^1 now:
       – Add a pair (l_i^1, l_j^2) as a match in PS (the final matched pairs set).
7. For letters that are still not resolved:
   - Compare the precedents environmental vectors of the template, PM_1, and the document, PM_2, in both directions, using f_1. Repeat step 6 with $PM1$, and PM_2.

- Compare the positions vectors of the template, P_1 and the document, P_2, in both directions, using f_1. Repeat step 6 with P_1, and P_2.

8. If there is only one unidentified letter l_k^2 left in the document alphabet:

   - Match it to the remaining letter in the language alphabet.

3 Experimental Results

We checked our methods on various types of texts in three languages: Hebrew, Russian, and English. The sources types in Hebrew experiments were on-line newspapers (HN, HN1, HN2), the Parliament protocols (HS), which represent the spoken language, several on-line scientific journals (HSJ) [5], and the Bible (HB). The English sources included Computer Science text (EC) [16], Spoken language (ES) [17], and the complete works of William Shakespeare (EL) [18]. The Russian corpus contained online newspapers (RN), scientific articles collection (RS), and prose of A. S. Pushkin's books (RP), and F. M. Dostojevsky (RD) [14].

We also examined the influence of the text size on the computed statistics in order to find the lower bound on the new documents size. We ran the algorithms on texts of sizes varying from 200Bytes (30-40 words) to 100 MB ($\sim$15 million words). Significant changes occur below 10K ($\sim$1,500 words), mostly in the bottom half of this range; the difference between 5K and 10KB was of 2-3 letters. Starting from 10K the matching results never changed (as shown in fig. 3.1).

Figures 3.2, 3.3, 3.4 are some illustrative graphs for different stages of the algorithm . The comparative results of the described algorithms are detailed in Table 2.

Both vector methods in isolation worked fine for homogeneous corpora, but produced some mismatching for different types of text. The best case is therefore when we compared two similar sources, such as two newspapers, the same author's books or two halves of the same source, such as the Bible, that was divided into two parts and matched one to the other. The worst case is when comparing ancient text to modern one, or written to spoken language. The lower hit ratio for spoken language samples can be explained by their high number of participants and since spoken language has almost no norms, or restrictions.

Still, there are quite a few "*strongly characterized letters*" that never produce misses as shown in Table 1. Assuming the constant alphabetic order of the encoding tables they enable us to determine the rest of the alphabet by the distances between letters in the alphabet, thus bringing the method to 100% accuracy even in the worst case. However we would like to stress that the worst case is practically very uncommon, since usually people view

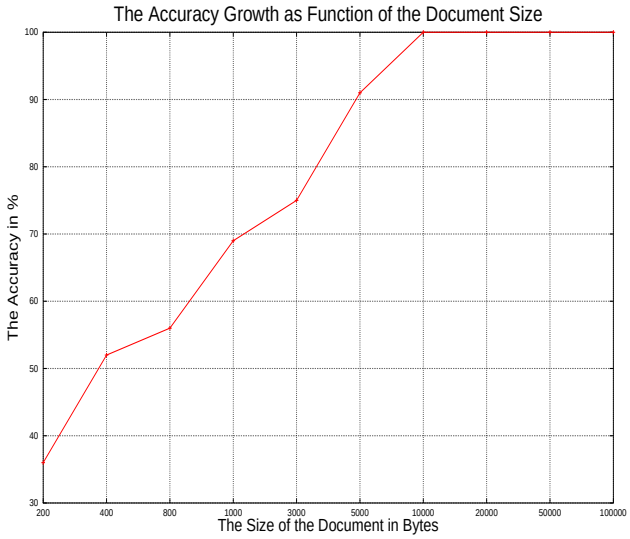

Figure 3.1: The document size has a crucial influence on the accuracy of the statistical alphabet recognition. The larger the document is the more accurate results are achieved. Naturally, the accuracy is really low for extremely small documents (of 30-150 words). It grows steadily with increasing text size, and reaches a maximum accuracy in texts 700-1,500 words in length.

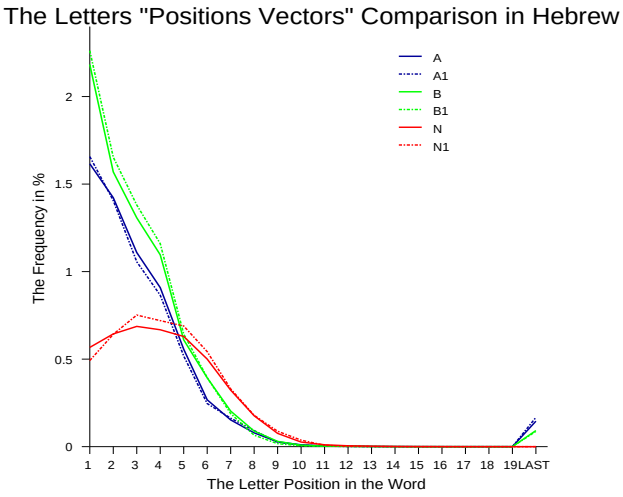

Figure 3.2: Example of identification of 3 letters that have similar frequencies, based on position vectors. The vectors are for A, B, N, obtained from two different texts in Hebrew: Scientific Journals (HSJ in Table 2) vs. Newspapers articles (test no. 4 in Table 2). Despite the similar frequencies, there is no ambiguity in matching by position vectors: the two vectors for each letter are nearly overlapping. Note that N never occurs at the last position since it has a special character for the final form.

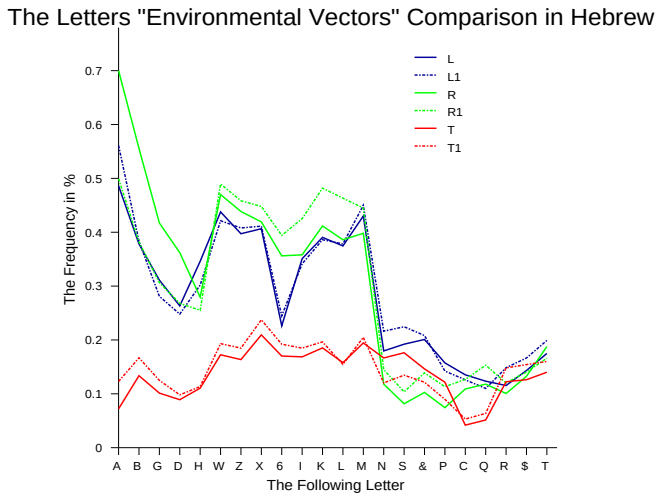

Figure 3.3: Matching three medium-frequency letters from Hebrew Scientific Journals (HSJ in Table 2) vs. Newspapers articles (test no. 4 in Table 2), using environmental vectors. These vectors consist of over 20 meaningful points (dimensions) while in the positions vectors usually only the first 10 and the last one were informative, therefore the similarity of the corresponding letters vectors is even more distinct in this model.

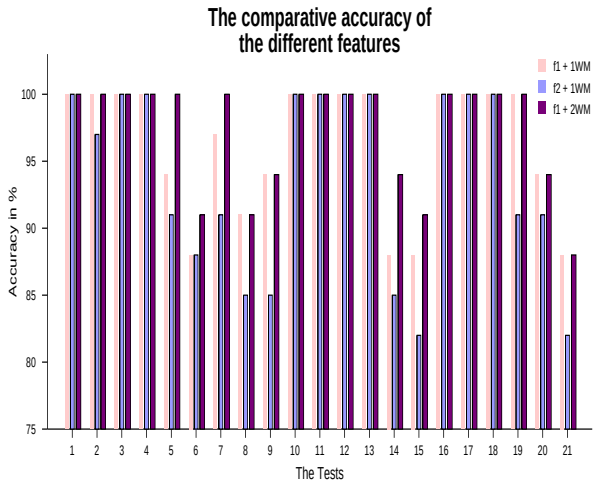

Figure 3.4: Comparison of the accuracy achieved by 3 online matching schemes, based on different combinations of the vector distance norm f_1 vs. f_2 and the use of one-way or two-way mapping (denoted '$1WM$' or '$2WM$'). The tests numbers refer to Table 2. As we can see, the $f1 + 2WM$ combination is always better than or equal to the other two.

Sources	Accuracy in %		
Compared Types	Positions Vectors	Environmental Vectors	Combined Method
HN-HN***	100	100	100
HN1-HN2*	91(2,1)**	100	100
HSJ-HSJ	100	100	100
HSJ-HN	100	100	100
HS-HS	94(2,0)	94(2,0)	100
HB-HB	100	100	100
HS-HN	76(6,2)	91(2,1)	91(2,1)
HB-HS	76(8,0)	91(3,0)	91(3,0)
HB-HSJ	76(6,2)	94(2,0)	94(2,0)
EC-EC	100	100	100
ES-ES	100	100	100
EL-EL	100	100	100
EL-ES	94(0,2)	94(2,0)	100
EL-EC	94(2,0)	94(2,0)	94(2,0)
ES-EC	76(4,4)	85(3,2)	91(1,2)
RN-RN	100	100	100
RS-RS	100	100	100
RP-RP	100	100	100
RD-RD	100	100	100
RP-RD	91(3,0)	91(2,1)	94(2,0)
RN-RP	80(5,2)	88(4,0)	88(4,0)

Table 2: The Final Results Table.
* We denote HN1 - HN2 for comparison of two different newspapers.
** The numbers of the errors of two types: (i) multiple matches (including the correct one), and (ii) unmatched letters, are shown in the parenthesis, respectively.
*** We denote X - X for comparison of two distinct parts of the same source.

Heb	D	H	X	I	L	M	N	k	m	n
Eng	A	B	C	D	E	F	G	L	M	N
Rus	B	V	D	E	Zh	I	J	K	L	M

Table 1: The "***strongly characterized letters***" lists in Hebrew, Russian and English. We used the Latin transliteration to represent Hebrew and Russian alphabet as described in [15], [13].

and download scientific and news articles, which both belong to the written-modern language style.

4 Conclusions and Future Work

We developed and presented a purely automatic method for alphabet recognition, based on a vector-space model. It produced fine results for languages from different families: Semitic, Slavic, and Indo-European. The time complexity of the vectors generation process is $\Theta(n)$, where n is a

number of characters in the new document. The matching procedure performance is bound by $\Theta(\mid Alphabet \mid^3)$.

We would like to suggest some possible extensions and further applications of our research. The method may be naturally used to easily identify the language of the document, by comparing its statistics to pre-computed templates of given languages. Only one of them will have a relatively small number of unmatched or multiple-matched letters. It can also be applied to determine documents direction (*'ltr', 'rtl'*) and representation type (*visual, logical*) by simply running the "*positions vectors*" algorithm in both ways and comparing the results to the templates. Obviously, only one of the two obtained vectors sets will match the templates, which reveals the correct direction and representation of the text. These are common difficulties since people tend not to supply this information despite the fact that it is mandated by the HTML 4 standard. In summary, the only thing our algorithm needs to receive in order to recognize the alphabet of a given piece of text is the list of candidate languages to choose from. Given this, it executes the three following stages, first identifies the language, then the representation type and the direction, and finally the character set.

References

[1] Bookstein A. and Klein S.T., Compression, Information Theory and Grammars: A Unified Approach, ACM Trans. on Information Systems 8 pp. 27-49, 1990.

[2] Bracewell M. and Karp D. A., O'Reilly Utilities – Quick Solutions for Windows 98 Annoyances, O'Reilly & Associates, Inc., 1998.

[3] Cormack G.V., and Horspool R.N., Data Compression using Dynamic Markov Modelling, Computer Journal 30:6 pp. 541-550, 1987.

[4] Graham I. S., HTML 4.0 Sourcebook. Wiley Computer Publishing, New York, pp. 450–451, 1998.

[5] Hebrew resources, http://www.snunit.k12.il/.

[6] Horspool R.N. and Cormack G.V., Dynamic Markov Modelling - A Prediction Algorithm, Proc. 19th Hawaii International Conference on System, Sciences Vol. II, pp. 700-707, 1986.

[7] Information Technology, ISONET Manual, ISO/IEC 8859, Jersey City, N.J., 1998.

[8] Jaeger G., Some Notes on the Formal Properties of Bidirectional Optimality Theory, to appear in Journal of Logic, Language, and Information, 2002.

[9] IBM Character Data Representation Architecture, Reference and Registry, SC09-2196-00, Dec. 1996.

[10] Kennedy B. and Musciano C., HTML & XHTML: The Definitive Guide, O'Reilly & Associates, Inc., 4th Edition, section 15.1, 2000.

[11] Northrup A., Introducing Microsoft Windows2000 Server, Microsoft Press, Washington, pp. 15–16, 1999.

[12] Smith B., SUN Microsystems Unveils Netscape 6 for Solaris, Sun's Press Releases, Brookline, MA, 2001.

[13] Russian transliteration, www.geocities.com/Colosseum/Track/7635/

[14] Russian resources, http://ruslit.virtualave.net.

[15] Segal, E., Itai A., Hebrew transliteration, http://www.cs.technion.ac.il/~erelsgl/bxi/hmntx/teud.html

[16] The Spoken English resource, http://www.athel.com

[17] The Scientific English resource, http://citeseer.nj.nec.com/cs

[18] English Literature resources, http://www.chemicool.com/

[19] Wiseman Y., Parallel Compression, Ph.D. Thesis, Computer Science Dept., Bar-Ilan University, Ramat-Gan, Israel, pp. 76-79, 2000.

[20] Yoon H. S., Soh J., Min B. and Yang H. S., Recognition of Alphabetical Hand Gestures Using Hidden Markov Model, IEICE Transactions Fundamentals, Vol. E82-A, No. 7, pp. 1358-1366, 1999.

[21] Ziv J. and Lempel A., Compression of Individual Sequences Via Variable-Rate Coding, IEEE Trans. on Information Theory IT-24 pp. 530–536, 1978.