THE EXTENDED BOYER-MOORE-HORSPOOL ALGORITHM FOR LOCALITY-SENSITIVE PSEUDO-CODE

Kengo Terasawa, Toshio Kawashima
Department of Media Architecture, Future University Hakodate, Hakodate, Japan

Yuzuru Tanaka
Meme Media Laboratory, Hokkaido University, Sapporo, Japan

Keywords: Information retrieval, String matching, Document image indexing.

Abstract: Boyer-Moore-Horspool (BMH) algorithm is known as a very efficient algorithm that finds a place where a certain string specified by the user appears within a longer text string. In this study, we propose the Extended Boyer-Moore-Horspool algorithm that can retrieve a pattern in the sequence of real vectors, rather than in the sequence of the characters. We reproduced the BMH algorithm to the sequence of real vectors by transforming the vectors into pseudo-code expression that consists of multiple integers and by introducing a novel binary relation called 'semiequivalent.' We confirmed the practical utility of our algorithm by applying it to the string matching problem of the images from "Minutes of the Imperial Diet," to which optical character recognition does not work well.

1 INTRODUCTION

Large and increasing amounts of information are accumulating on the internet. In tandem with this, the importance of information retrieval technology, which enables users to extract necessary information efficiently from the available information, is also increasing. String matching problem is one such basic technology that is intensively studied. In particular, the Boyer-Moore (BM) algorithm (Boyer and Moore, 1977) and Boyer-Moore-Horspool (BMH) algorithm (Horspool, 1980) are extremely efficient algorithms that run in sublinear time, i.e. it is possible to retrieve the keyword string without accessing the whole text string.

The purpose of this study is to broaden the retrieval target from a sequence of characters to a sequence of real vectors. This technology can provide a platform where it is feasible to conduct high-speed similarity searches on a variety of time-series or pseudo- time-series data.

The main element of the technology in this extension addresses two points. One is LSPC (Locality-Sensitive Pseudo-Code) (Terasawa and Tanaka, 2007a), which is a kind of discretization of real vectors. The LSPC discretize vectors into a set of integers with less loss of information compared with usual vector quantization. However, because a binary relation 'semiequivalent' used in LSPC is not an equivalence relation and does not satisfy the transitive law, existing string matching algorithms are not always applicable to LSPC. Therefore, this study also introduces a second element, i.e. we arranges BMH algorithms to be applicable to LSPC. As stated in the title of the paper, this is termed an Extended Boyer-Moore-Horspool Algorithm.

2 OUTLINE OF LSPC

Locality-Sensitive Pseudo-Code (LSPC) is a technology where real vectors can be converted into pseudo-code expressions without significant loss of their information. In this section we introduce the outline of LSPC. A more detailed description can be found in (Terasawa and Tanaka, 2007a).

Figure 1 illustrates an example of LSPC. The black circles represent vectors distributed in the $\mathbb{R}^d$ space, and the three-tuples of integers in brackets, like (8,1,4), represent the pseudo-codes that are the result of the discretization of the vectors. In this manner, allocating multiple integers to one vector is the main characteristic of LSPC. Another characteristic

Terasawa K., Kawashima T. and Tanaka Y..
THE EXTENDED BOYER-MOORE-HORSPOOL ALGORITHM FOR LOCALITY-SENSITIVE PSEUDO-CODE.
DOI: 10.5220/0003369004370441
In *Proceedings of the International Conference on Computer Vision Theory and Applications* (VISAPP-2011), pages 437-441
ISBN: 978-989-8425-47-8

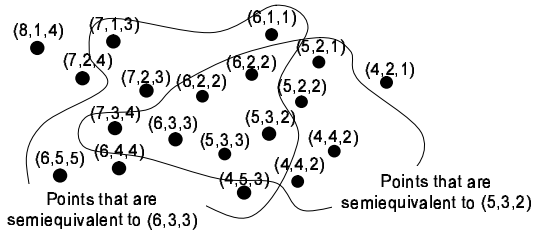


Figure 1: An example of LSPC.

of LSPC is a binary relation termed 'semiequivalent.' Two pseudo-codes are regarded to be semiequivalent if at least one of their elements takes the same value.

The above allows the following summary of the definitions of LSPC.

Definition 1. *For a d-dimensional vector p, LSPC $C(p)$ is given as d'-tuples of integers:*

$$p \in \mathbb{R}^d \mapsto C(p) = (c_1(p)\ c_2(p)\ \cdots\ c_{d'}(p))^T \in \mathbb{N}^{d'}.$$

Definition 2. *Two LSPC $C(p) = \{c_i(p)\}$ and $C(q) = \{c_i(q)\}$ are regarded to be semiequivalent iff $\exists i$ s.t. $c_i(p) = c_i(q)$.*

In composing the LSPC, we used the hash value obtained from the Locality-Sensitive Hashing (LSH) (Gionis et al., 1999; Datar et al., 2004; Andoni and Indyk, 2006; Terasawa and Tanaka, 2007b) as an element of LSPC.

LSPC has richer descriptive power compared with usual vector quantization. In usual vector quantization, there may be cases where different codes are allocated to similar vectors in the vicinity of boundaries. However, with LSPC, a range which is semiequivalent to a specific pseudo-code may overlap with a range which is semiequivalent to another pseudo-code (as shown in Fig. 1). As a result, the probability of two pseudo-codes to be semiequivalent is highly sensitive to the distance between two vectors (an example is shown in Fig. 2). Therefore, it is possible to estimate whether the distance between the two vectors is large or small by examining only whether the two pseudo-codes are semiequivalent.

Here it must be noted that the computational costs to examine whether pseudo-codes $C(p)$ and $C(q)$ satisfy semiequivalent relations is smaller than the cost to compute the distance between the two vectors p and q. This difference is significant especially when the vector space is of higher dimensions. Vectors used in pattern recognition often have a very high number of dimensions, making it very likely that using LSPC will reduce the computational costs.

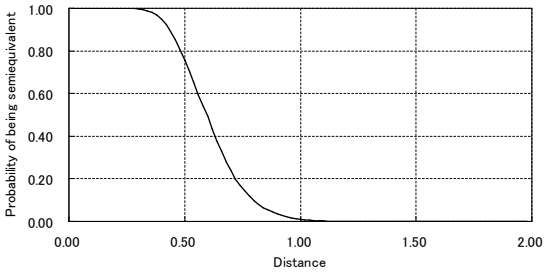


Figure 2: An example of the probability of two pseudo-codes to be semiequivalent with respect to the distance between two vectors.

3 THE BOYER-MOORE-HORSPOOL (BMH) ALGORITHM

This section will overview the BMH algorithm which is the basis for the Extended BMH algorithm.

First, let us accurately formulate the problem. The string matching problem is formulated as follows.

Definition 3 (The String Matching Problem)**.** *Given string P of length n (referred to as the pattern or keyword) and string T of length $m(> n)$ (referred to as text). The string matching problem is to search for all starting position i of a keyword in a text such that $P(\xi) = T(i+\xi-1)$ for all $\xi = 1,2,...,n$*

Here $P(\xi)$ represents the ξ-th character of P. For example, with $P = abab$ and $T = abxababababxababx$, the solution to the string marching problem is $\{4,6,11\}$. In the following, $S[i:j]$ $(j \geq i)$ represents the substring of S that starts from i-th character and ends at j-th character of S.

The naive method of string matching is a method in which matching is conducted on every substring of T with length n to pattern P. The computational cost with this method (frequency of character comparison) is $O(mn)$ at maximum.

Now, let us look at the BMH algorithm. The naive method conducts matching of strings by shifting the starting point of the substring T by one place for every trial. Compared with this, the BMH algorithm reduces computational cost by attempting to shift the starting point by more than one place.

Figure 3 illustrates an example of an execution of the BMH algorithm. Suppose $T = abcbaxabacabbc$ and $P = abac$. First, the BMH algorithm matches P and $T[1:4]$. The matching proceeds from right to left. In this example, the 4th character of $T[1:4]$ and the 4th character of P is compared, and the result is "not match." Then the next substring to be examined is $T[3:6]$, i.e., the starting point of the substring to be

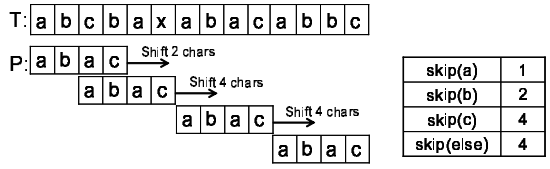


Figure 3: An example of BMH algorithm.

examined shifts two place because the last character of $T[1:4]$ is already known as *b*, it leads the fact that we do not need to examine $T[2:5]$.

The important point here is how many places we should shift the starting point of the substring of T. It only depends on 4th character of $T[i:i+3]$. Here, the value of "how many places to shift" will be expressed as "skip function." The function skip(x) is then defined as the number of places to shift when the last character of substring $T[i:i+3]$ is *x*, where the minimum value is 1 and the maximum value is n ($=4$, in this example).

The skip function is formed as follows. The skip function includes all the possible characters (referred to as alphabet Σ) as the domain and integer $[1,n]$ as its range. First, skip(x) is initialized to n for all $x \in \Sigma$. Next, if the $(n-i)$-th character of P is *x*, then skip(x) is updated to *i*. The skip function is obtained by iterating this procedure in the order of $i = n-1, n-2, \cdots, 1$.

4 THE EXTENDED BOYER-MOORE-HORSPOOL (BMH) ALGORITHM

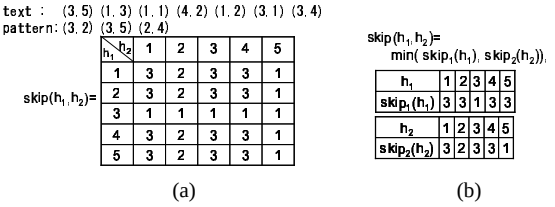
The objective of this study is to extend the BMH algorithm to LSPC, and in the following the definition of the problem identified in Definition 3 above is reorganized as a problem that has been extended to LSPC.

Definition 4 (The String Matching Problem Extended to LSPC)**.** *Given LSPC string P of length n and LSPC string T of length m*$(>n)$*. The extended string matching problem is to search for all starting position i of a keyword in a text such that* $P(\xi) \sim T(i+\xi-1)$ *for all* $\xi = 1,2,...,n$*,*

Here $S(i) \sim T(j)$ implies that $S(i)$ and $T(j)$ are semiequivalent.

The Extended BMH algorithm, generally, is the same as the BMH algorithm. Namely:

> [Extended BMH Algorithm]
> First, *i* is set to one and examine whether $P(\xi) \sim T(i+\xi-1)$ for all $\xi = 1,2,...,n$ can be satisfied. Similar to the original BMH algorithm, the matching proceeds from right to left. Next, to conduct matching for a new substring, the starting position *i* is moved by skip$(T(i+n-1))$. The algorithm will finish at $i+n-1 > m$.



(a) (b)

Figure 4: skip function in LSPC, (a): Array size $\mathcal{M}^{d'}$, (b): Array size $\mathcal{M}d'$.

The difference between the Extended BMH algorithm and the BMH algorithm is the composition of the skip function. In theory, the skip function for the Extended BMH algorithm is formed by the following process. The domain of the skip function includes all the values that could be obtained by the pseudo-code (following the example of the text string, we refer to this as alphabet Σ). First, skip(x) is initialized to n for all $x \in \Sigma$. Next, skip(x) is updated to *i* concerning all pseudo-code $x \in \Sigma$ s.t. $x \sim P(n-i)$. The skip function is obtained by iterating this procedure in the order of $i = n-1, n-2, ..., 1$ (Fig. 4(a)).

This process is theoretically possible but practically not. To utilize this method it requires array size that is proportional to the size of Σ to store the skip function; however, LSPC has d' integers as its element. If we assume the value of each element to be from 1 to $\mathcal{M}$, the array size will be $\mathcal{M}^{d'}$; therefore, when we utilize values of d' larger than tens or hundreds, huge memory requirement poses a problem. Thus, the following theorem becomes significant.

Theorem 1. *The skip function of the pseudo-code* $C(p) = (c_1(p)c_2(p)...c_{d'}(p))$ *can be expressed as follows:*

$$skip(C(p)) = \min(skip_1(c_1(p)), skip_2(c_2(p)), \ldots, skip_{d'}(c_{d'}(p)))$$

where $skip_i(c_i(p))$ *is the skip function that is formed by the same method as the traditional BMH algorithm.*

iProofj. When $\lambda < \min_{i=1,...,d'}(\text{skip}_i(c_i(p)))$, then $P(n-\lambda)$ and $C(p)$ will never be semiequivalent. This is because, when $P(n-\lambda) \sim C(p)$, there exists $\delta \in \{1,...,d'\}$ s.t. the δ-th element of $P(n-\lambda)$ and the δ-th element of $C(p)$ are equal; however, this contradicts $\lambda < \text{skip}_\delta(c_\delta(p))$. On the other hand, when $\lambda = \min_{i=1,...,d'}(\text{skip}_i(c_i(p)))$, then there exists δ s.t. $\lambda = \text{skip}_\delta(c_\delta(p))$, which means that the δ-th element of $P(n-\lambda)$ and the δ-th element of $C(p)$ are

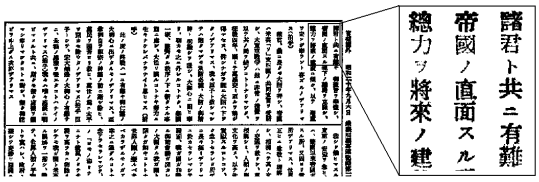
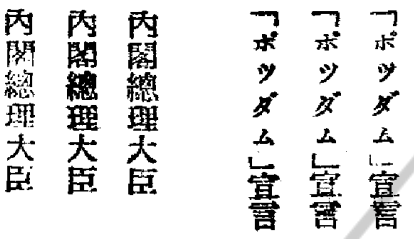


Figure 5: Images from the "Minutes of the Imperial Diet."



(a) (b)

Figure 6: Keywords used in the experiment. (a): Prime Minister (length=6, frequency=15), (b): Potsdam Declaration (length=8, frequency=11).

equal, i.e., $P(n-\lambda) \sim C(p)$. Thus, $\mathrm{skip}(C(p)) = \min_{i=1,\ldots,d'}(\mathrm{skip}_i(c_i(p)))$. □

By this theorem, the memory requirement is reduced from $\mathcal{M}^{d'}$ to $\mathcal{M}d'$, and the implementation becomes practically possible (Fig. 4(b)).

5 EXPERIMENT USING LOW-QUALITY DOCUMENT IMAGES

As an application of the Extended BMH algorithm, an experiment in string matching was executed using low-quality document images which are difficult to convert to text by OCR. We compared the computational costs with the extended BMH algorithm and the naive method. Note that the naive method is currently the only method that is applicable to LSPC, except for the proposed method.

5.1 Materials and Experimental Setup

The material used in our experiment was the "Minutes of the Imperial Diet" from National Diet Library, Japan [1]. This database contains the image data of the 28th (1911) – 92nd (1947) Imperial Diet sessions. For 88th (1945) – 92nd (1947), text data is also available. We used 32 images from 88th session. Sample page is displayed in Fig. 5. The resolution per character is approximately 50 pixels.

First, the document images are segmented to characters. Character segmentation was conducted by proprietary OCR software "MDTOCR v.6.0." Note that character segmentation is easier problem than character recognition problem. The accuracy of character segmentation was over 99% for the material images, while accuracy of character recognition is less than 85%.

Each segmented character image is converted to feature vector. As a feature vector, we utilized the gradient distribution feature (GDF) (Terasawa et al., 2006). The segmented image was divided into smaller 4×4 domains, which were used to form feature vectors of 128 dimensions, just similar to the method utilized in SIFT (Lowe, 2004).

Next, each feature vector was converted into LSPC. The LSH family used was the SLSH-orthoplex (Terasawa and Tanaka, 2007b) because the feature vectors of GDF is always normalized to a unit length. The parameters used in LSPC conversion was determined by preliminary experiment.

Thus, the document images were converted to the sequence of LSPC and were ready to be conducted the string matching based on Extended BMH algorithm.

5.2 Experiment

The total number of characters of the document images used in this experiment was 64900. For this experiment, we had selected two keywords from the whole document, as shown in Fig. 6. Keyword (a) means "Prime Minister" in Japanese, having six character length and appearing 15 times in the document. Keyword (b) means "Potsdam Declaration" in Japanese, having eight character length and appearing 11 times in the document. For both keyword, each occurrences was used as a query, using both the naive method and the Extended BMH algorithm.

5.3 Experimental Results

The result is summarized in Table 1. In the table, (#comp) represents how many times the character to character comparison were executed and (#skip) represents how many times the skip function were evaluated. Displayed digits are mean values over each occurrences was used as a query. Note that non-perfect recall and precision ratio comes from two reasons. One reason is that there exists deformation of the characters in the image, and another reason is the nature of randomized algorithm. Since the transformation of feature vectors into LSPC is a randomized algorithm, the accuracy of the LSPC-based retrieval

[1] http://teikokugikai-i.ndl.go.jp/

Table 1: Experimental Results.

Keyword	Naive	Extended BMH		Recall	Precision
	(#comp)	(#comp)	(#skip)	(%)	(%)
Prime Minister	79127.8	21973.2	17649.6	88.57	80.87
Potsdam Declaration	78833.6	16850.2	13488.0	61.82	100.00

(number of characters = 64900)

is not assured to be perfect. Although it is possible to improve the accuracy by increasing the trial frequency or adjusting other parameters, this will involve a trade-off with the computational complexity. Those are the nature of randomized algorithms.

As illustrated by the tables, the number of times of character to character comparison of the Extended BMH algorithm is lower than the naive method for both keywords. Moreover, it is lower than the total number of characters (64900), which means that retrieval in sublinear time was accomplished.

However, the number of times of evaluation of the skip function must be considered as an additional cost for the naive method with the Extended BMH algorithm. Even though one 'comp' process and one 'skip' process is not exactly equal — the former is a process that evaluates the match/non-match of d' integers, while the latter is a process that obtains the minimum among d' integers — we can roughly estimate the cost of the Extended BMH algorithm just adding (#skip) and (#comp). In this case, also, we can conclude that the computational costs of the Extended BMH is reduced to below that of the naive method.

6 CONCLUSIONS

In this paper we have proposed an Extended BMH algorithm, which was developed from the BM and BMH algorithm, to allow searching for specific strings based on a sequence of real vectors. As an example of its application, an experiment with string matching using low-quality document images where optical character recognition does not work well was performed. The results showed that the algorithm can contribute to reducing the computational cost compared with the naive method.

Our future work will focus on developing an efficient algorithm to realize Inexact Matching rather than Exact Matching. With such an advanced algorithm, it would be possible to develop a fast algorithm using LSPC, which is applicable to more difficult problems such as string matching of handwritten documents.

REFERENCES

Andoni, A. and Indyk, P. (2006). Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. In *Proc. Symposium on Foundations of Computer Science, FOCS'06, pp. 459–468*.

Boyer, R. S. and Moore, J. S. (1977). A fast string searching algorithm. In *Communications of the ACM, vol. 20, pp. 762–772*.

Datar, M., Indyk, P., Immorlica, N., and Mirrokni, V. (2004). Locality-sensitive hashing scheme based on p-stable distributions. In *Proc. 20th ACM Symposium on Computational Geometry, SoCG2004, pp. 253–262*.

Gionis, A., Indyk, P., and Motwani, R. (1999). Similarity search in high dimensions via hashing. In *Proc. 25th Int. Conf. on Very Large Data Base, VLDB1999, pp. 518–529*.

Horspool, R. N. (1980). Practical fast searching in strings. In *Software – Practice & Experience, vol. 10, issue 6, pp. 501–506*.

Lowe, D. G. (2004). Distinctive image features from scale-invariant keypoints. In *International Journal of Computer Vision, vol. 60, no. 2, pp. 91–110.*

Terasawa, K., Nagasaki, T., and Kawashima, T. (2006). Improved handwritten text retrieval using gradient distribution features (written in japanese). In *Proc. Meeting on Image Recognition and Understanding, MIRU2006, pp.1325–1330*.

Terasawa, K. and Tanaka, Y. (2007a). Locality sensitive pseudo-code for document images. In *Proc. 9th Int. Conf. on Document Analysis and Recognition, ICDAR2007, vol. 1, pp. 73–77*.

Terasawa, K. and Tanaka, Y. (2007b). Spherical lsh for approximate nearest neighbor search on unit hypersphere. In *Proc. 10th Workshop on Algorithms and Data Structures, WADS2007, LNCS4619, pp. 27–38*.