Knuth-Morris-Pratt (KMP) and Boyer-Moore Algorithm Implementation in Digital Image Processing Template Matching

Annisaur Rosi Lutfiana 13512088[1]
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]*annisaurrosi@s.itb.ac.id*

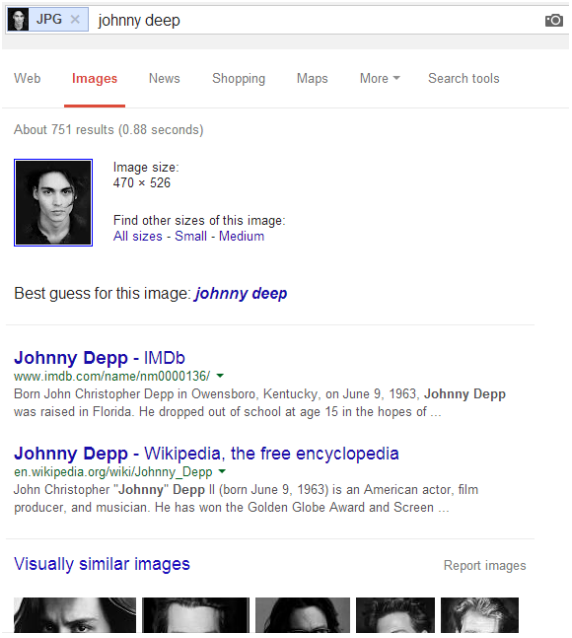
***Abstract*— To find out the source of the image which is initially unknown for various purposes becomes a lot easier for today, one of the example is by using the Google feature "search by image". The technique used by the application and other applications which requires searching or matching techniques is representing a derivation of string matching algorithms, as known as template matching. The explanation in this paper includes the brief description of template matching and its applications in digital image processing, also how it is implemented in two widely known string matching algorithms, Knuth-Morris-Pratt (KMP) and Boyer-Moore string matching algorithm.**

***Index Terms*—Template matching, string matching, digital image processing, content-based image retrieval**

I. Introduction

In this information era, the usages of digital images are almost never separated to everyday life. There are times when the usage of digital images containing specific information that unable to be extracted into information in the more general forms, such as numbers or certain strings. This problem most likely to happen because the digital image is a form of data that is extremely rich in its properties, such as size, color, texture, shape, and other components. The acts of identifying, searching, and matching of digital images which are parts of the digital image processing will be much more difficult than the processing of data or strings.

One of the problems that most probably occurred in the usage of digital images is, that if the source of the digital image is unknown. If someone wants to find the source of a digital image, retrieve its informations, or get the access for other digital images which are similar to the current image is quite a difficult problem to be solved and is not as easy as doing the same thing to a text. Fortunately, we can find a new feature in Google to solve these problems, which known as the "search by image". It is relatively very easy for users to identify images simply by providing a digital image to be processed without the need to describe the digital images as words and search for it in regular Google search.



Picture 1.1 Google search by image

Another type of digital image processing which can be considered much more important than everyday common identification of digital image is for example, facial recognition or recognition of biological features such as iris or fingerprint. This process is typically used to access particular restrictions, which can be functioned as a key for a

room or anything. It has been proven that the use of such recognition is much more secure than a password or manual physical key, to remember that passwords and keys can be lost or stolen. The program will store the data of user's features who are granted for access, so the program can match the image retrieved by its sensor with the pattern data provided.

There are a lot of other programs or features which implementing the matching process of the digital images. Some of them includes the access of intellectual property in the form of digital images, automatic discovery and face tags that we can find in Google Picasa and Facebook applications, the atomatic nudity censorship in digital images, video, and other media, and many other applications. Of course, digital image processing is needed in those fields that unable to be separated from the usage of digital images, such as photography, art, medical, even in law enforcement.

Digital image processing with the functions mentioned above is implementing the technique called Content-Based Image Retrieval (CBIR). This technique uses template matching algorithm, which is a derivation of the string matching algorithm. Known string matching algorithms include algorithms Knuth-Morris-Pratt (KMP) and the Boyer-Moore.

II. Related Theories

2.1 String Matching

String matching is the problem of finding occurrence(s) of a pattern string within another string or body of text. There are many different algorithms for efficient searching. Also known as exact string matching, string searching, or text searching.[1]

Here is the known string matching algorithms:

Single pattern algorithms

1. Naïve string matching algorithm
2. Rabin–Karp string matching algorithm
3. Finite-state automaton based search
4. Knuth–Morris–Pratt algorithm
5. Boyer–Moore string matching algorithm
6. Bitap algorithm (*shift-or*, *shift-and*, *Baeza–Yates–Gonnet*)

Algorithms using a finite set of patterns

1. Aho–Corasick string matching algorithm
2. Commentz-Walter algorithm
3. Rabin–Karp string matching algorithm[2]

2.2 Template Matching

Template matching is a technique in digital image processing for finding small parts of an image which match a template image. It can be used in manufacturing as a part of quality control, a way to navigate a mobile robot, or as a way to detect edges in images.[3] A basic method of template matching uses a convolution mask (template), tailored to a specific feature of the search image, which we want to detect. This technique can be easily performed on grey images or edge images. The convolution output will be highest at places where the image structure matches the mask structure, where large image values get multiplied by large mask values.

This method is normally implemented by first picking out a part of the search image to use as a template: We will call the search image **S(x, y)**, where **(x, y)** represent the coordinates of each pixel in the search image. We will call the template **T(x_t, y_t)**, where **(x_t, y_t)** represent the coordinates of each pixel in the template. We then simply move the center (or the origin) of the template **T(x_t, y_t)** over each **(x, y)** point in the search image and calculate the sum of products between the coefficients in **S(x, y)** and **T(x_t, y_t)** over the whole area spanned by the template. As all possible positions of the template with respect to the search image are considered, the position with the highest score is the best position. This method is sometimes referred to as 'Linear Spatial Filtering' and the template is called a filter mask.

For example, one way to handle translation problems on images, using template matching is to compare the intensities of the pixels, using the **SAD** (Sum of absolute differences) measure.

A pixel in the search image with coordinates **(x_s, y_s)** has intensity **$I_s(x_s, y_s)$** and a pixel in the template with coordinates **(x_t, y_t)** has intensity **$I_t(x_t, y_t)$**. Thus the absolute difference in the pixel intensities is defined as **$\mathrm{Diff}(x_s, y_s, x_t, y_t) = |I_s(x_s, y_s) - I_t(x_t, y_t)|$**.

$$SAD(x,y) = \sum_{i=0}^{T_{\text{rows}}} \sum_{j=0}^{T_{\text{cols}}} \mathrm{Diff}(x+i, y+j, i, j)$$

The mathematical representation of the idea about looping through the pixels in the search

image as we translate the origin of the template at every pixel and take the SAD measure is the following:

$$\sum_{x=0}^{S_{\text{rows}}} \sum_{y=0}^{S_{\text{cols}}} SAD(x,y)$$

S$_{\text{rows}}$ and **S**$_{\text{cols}}$ denote the rows and the columns of the search image and **T**$_{\text{rows}}$ and **T**$_{\text{cols}}$ denote the rows and the columns of the template image, respectively. In this method the lowest SAD score gives the estimate for the best position of template within the search image. The method is simple to implement and understand, but it is one of the slowest methods.[4]

In this simple implementation, it is assumed that the above described method is applied on grey images: This is why **Grey** is used as pixel intensity. The final position in this implementation gives the top left location for where the template image best matches the search image.

```
minSAD = VALUE_MAX;

// loop through the search image
for ( int x = 0; x <= S_rows - T_rows;
x++ ) {
    for ( int y = 0; y <= S_cols -
T_cols; y++ ) {
        SAD = 0.0;

        // loop through the template
image

        for ( int j = 0; j < T_cols; j++
)
            for ( int i = 0; i < T_rows;
i++ ) {

                pixel p_SearchIMG =
S[x+i][y+j];
                pixel p_TemplateIMG =
T[i][j];

                SAD += abs(
p_SearchIMG.Grey - p_TemplateIMG.Grey );
            }

        // save the best found position
        if ( minSAD > SAD ) {
            minSAD = SAD;
            // give me min SAD
            position.bestRow = x;
            position.bestCol = y;
            position.bestSAD = SAD;
        }
    }
}
```

2.3 Knuth-Morris-Pratt (KMP) Algorithm

Knuth-Morris-Pratt string matching algorithm (or KMP algorithm) searches for occurrences of a "word" W within a main "text string" S by employing the observation that when a mismatch occurs, the word itself embodies sufficient information to determine where the next match could begin, thus bypassing re-examination of previously matched characters.[5]

```
input:
an array of characters, S(the text to be
searched)
an array of characters, W(the word sought)
output:
an integer(the zero - based position in S
at which W is found)

define variables :
an integer, m ← 0 (the beginning of the
current match in S)
an integer, i ← 0 (the position of the
current character in W)
an array of integers, T(the table,
computed elsewhere)

while m + i < length(S) do
if W[i] = S[m + i] then
if i = length(W) - 1 then
return m
let i ← i + 1
else
if T[i] > -1 then
let i ← T[i], m ← m + i - T[i]
else
let i ← 0, m ← m + 1

(if we reach here, we have searched all of
S unsuccessfully)
return the length of S
```

2.4 Boyer-Moore Algorithm

Boyer–Moore string search algorithm is an efficient string searching algorithm that is the standard benchmark for practical string search literature.[6] The algorithm preprocesses the string searched for (the pattern) but not the string being searched in (the text). It is thus well-suited for applications in which the pattern is much shorter than the text or does persist across multiple searches. The Boyer-Moore algorithm uses information gathered during the preprocess step to skip sections of the text, resulting in a lower constant factor than many other string algorithms. In general, the algorithm runs faster as the pattern length increases. The key feature of the algorithm is to match on the tail of the pattern rather than

the head, and to skip along the text in jumps of multiple characters rather than searching every single character in the text.

```
input:
String T(text with n characters and
P(pattern) with m characters
output:
Starting index offirst substring of T
matching p, or an indication that P is not
a substring of T
i <--m - 1
j <--m - 1

repeat
if P[j] = T[i] then
        if j = 0 then
                return i
        else { check next character }
        i <--i - 1
        j <--j - 1
else { P[j] <> T[i] move the pattern }
i <--i + m - j - 1
i <--i + max(j - last(T[i]), match(j))
j <--m - 1
until i > n - 1
return "There is no substring of T
matching P."
```

2.5 Content-Based Image Retrieval

Content-based image retrieval (CBIR), also known as query by image content (QBIC) and content- based visual information retrieval (CBVIR) is the application of computer vision techniques to the image retrieval problem, that is, the problem of searching for digital images in large databases. Content - based image retrieval is opposed to traditional concept-based approaches.[7]

Initial CBIR systems were developed to search databases based on image color, texture, and shape properties. After these systems were developed, the need for user-friendly interfaces became apparent. Therefore, efforts in the CBIR field started to include human-centered design that tried to meet the needs of the user performing the search. This typically means inclusion of: query methods that may allow descriptive semantics, queries that may involve user feedback, systems that may include machine learning, and systems that may understand user satisfaction levels.

III. Implementations and Experiments

Hereby the C implementation of Knuth-Morris-Pratt string matching algorithm for template matching, using the same properties as the example of template matching in section 2.2.

```
minSAD = VALUE_MAX;

// loop through the search image

for (int x = 0; x <= S_rows - T_rows; x++) {
        for (int y = 0; y <= S_cols -
T_cols; y++) {
                SAD = 0.0;

                // construct the lookup
table

                T =
(int*)malloc(sizeof(T_rows)*
sizeof(T_cols));
                T[0] = -1;
                for (int x = 0; x <= S_rows
- T_rows; x++) {
                        T[x + 1] = T[x] +
1;
                        while (T[x + 1] > 0
&& p_TemplateIMG[x] != p_TemplateIMG[T[x +
1] - 1])
                                T[x + 1] =
T[T[x + 1] - 1] + 1;
                }

                // template matching

                for (x = y = 0; S[x] !=
'\0';) {
                        if (y < 0 || S[x]
== T[y]) {
                                ++x, ++y;
                                if (T[y] ==
'\0') {

        result = text + x - y;

        break;
                                }
                        }
                        else x = T[y];
                }

                for (int j = 0; j < T_cols;
j++)
                for (int i = 0; i < T_rows;
i++) {

                        pixel p_SearchIMG =
S[x + i][y + j];
                        pixel p_TemplateIMG
= T[i][j];

                        SAD +=
abs(p_SearchIMG.Grey - p_TemplateIMG.Grey);
                }

                // save the best found
position
                if (minSAD > SAD) {
                        minSAD = SAD;
                        // give me min SAD
                        position.bestRow =
x;
```

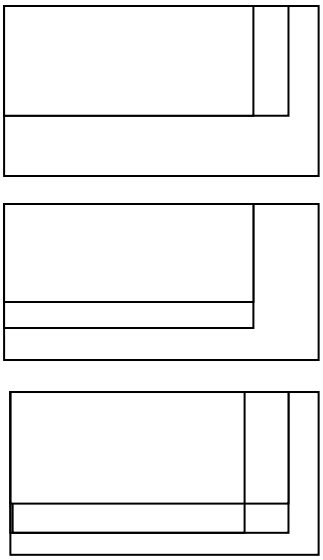
```
                        position.bestCol =
y;
                        position.bestSAD =
SAD;
                }
        }
}
```

Below written the Boyer-Moore implementation of the same functions and properties as the template matching example.

```
minSAD = VALUE_MAX;

// loop through the template image

for (int x = 0; x <= S_rows - T_rows; x++) {
        for (int y = 0; y <= S_cols - T_cols;
y++) {
                SAD = 0.0;

                // initialize skip lookup
table
                for (x = 0; x < 256; i++)
                        skipTable[i] =
p_TemplateIMG;

                p_TemplateIMG = P;

                // decrease PLength here to
make it an index
                i = --PLength;

                do
                {

        skipTable[*p_TemplateIMG++] = i;
                } while (i--);

                lastChar = *--p_TemplateIMG;

                // start p_TemplateIMG,
position pointer at possible end of image
                p_TemplateIMG = T + PLength;
                TLength -= PLength + (PLength
- 1);

                while ((int)TLength > 0)
                {
                        unsigned int skip;

                        skip =
skipTable[*p_TemplateIMG];
                        p_TemplateIMG +=
skip;
                        TLength -= skip;
                        skip =
skipTable[*p_TemplateIMG];
                        p_TemplateIMG +=
skip;
                        TLength -= skip;
                        skip =
skipTable[*p_TemplateIMG];

                        if (*p_TemplateIMG !=
lastChar) /*if (skip > 0)*/
                        {
                                // image does
not match, realign image and try again
                                p_TemplateIMG
+= skip;
                                TLength -=
skip;
                                continue;
                        }

                        // we had a match, we
could be at the end of the image
                        i = PLength;

                        do
                        {
                                // Have we
found the entire Ping?
                                if (i-- == 0)

        return p_TemplateIMG;
                        } while (*--
p_TemplateIMG == Ping[i]);

                        // skip past the part
of the Ping that we scanned already
                        p_TemplateIMG +=
(PLength - i + 1);
                        TLength--;
                }

                // we reached the end of the
T, and didn't find the Ping
                return NULL;
        }

        // save the best found position
        if (minSAD > SAD) {
                minSAD = SAD;
                // give me min SAD
                position.bePow = x;
                position.bestCol = y;
                position.bestSAD = SAD;
        }
}
}
```

Both of the algorithms (KMP and Boyer-Moore) are considered to be not very effective and get through many difficulties to solve problems through the search and matching two-dimensional, as required for template matching as a method of use of CBIR. Which is considered as the most effective algorithm for two-dimensional search algorithm which are Polcar, the following explanations and illustration (for string matching).

a. For each prefix of the text *A*, we compute the set of suffixes of *A* that are also a prefix of the patterm: ***suff(A) ∩ pref(P)***



Picture 1.2 Illustration of suffix and prefix in Polcar algorithm

b. In derivations of the corresponding 1D pattern matching algorithms, sets of prefixes of the pattern are represented by their element of maximum length.
c. In 2D there is not always one unique maximum but these sets of matrices can be represented by their maximal elements.

For the comparation between KMP and Boyer-Moore itself, considered that both of the algorithms has its own specialties depending on the problem and situations. The Knuth-Morris-Pratt (KMP) algorithm is a good choice if we need to search for the same pattern repeatedly in many different queries, but only works best in short pattern applications. Therefore, for this template matching problem which implementing the usage of digital images processing as the objects that relatively large and complex in terms of size and properties, Boyer-Moore is a better algorithm for this problem in its application.

Here is the complexity result of both algorithms, in general pattern matching application.

Algorithm	**Pre-processing time**	**Matching time**
Knuth-Morris-Pratt algorithm	$\Theta(m)$	$\Theta(n)$
Boyer-Moore algorithm	$\Theta(m + \|\Sigma\|)$	$\Omega(n/m)$, $O(nm)$

IV. Conclusion

Template matching is a method known as a derivation from a string or pattern matching method, which is implemented in the Content-Based Image Retrieval (CBIR), as part of the application of digital image processing, one of those vital and important features in the usage of software. Although it is considered that two algorithms we're working on (Knuth-Morris-Pratt and Boyer-Moore) are less effective in the solving attempt of two-dimensional pattern matching problem, both of these algorithms are considered to represent the issue as its own form of settlement.

KMP and Boyer-Moore algorithm has their own specialties when it comes to solve various kind of template matching problems, but Boyer-Moore is considered to be the most effective template matching algorithm for this kind of problem because of its adeptness in handling large size of template, or in this term, digital images.

VII. Acknowledgment

Annisa'ur Rosi Lutfiana, as the author of this paper, want to express her grateful thanks to God for His blessing during the making of this paper, her deepest gratitude to Dr. Ir. Rinaldi Munir, M.T. and Masayu Leylia Khodra, ST., MT. as the lecturers of IF2211 – Strategi Algoritma.

Special thanks to my family back home and all my friends in HMIF for all the supports and guidances. Also thanks for all of the authors which I got a lot of this study references from, may God bless your knowledge with endless abundances.

References

[1] Paul E. Black, in *Dictionary of Algorithms and Data Structures*, Vreda Pieterse and Paul E. Black, ed. 5 May 2005.

[2] Melichar, Borivoj, Jan Holub, and J. Polcar. *Text Searching Algorithms*. Volume I: Forward String Matching. Vol. 1. 2 vols., 2005.

[3] R. Brunelli, *Template Matching Techniques in Computer Vision: Theory and Practice*, Wiley, ISBN 978-0-470-51706-2, 2009

[4] B. Sirmacek, C. Unsalan. *Urban Area and Building Detection Using SIFT Keypoints and Graph Theory*, IEEE Transactions on Geoscience and Remote Sensing, Vol.47 (4), pp. 1156-1167, April 2009.

[5] Knuth, Donald; Morris, James H., jr; Pratt, Vaughan (1977). *SIAM Journal on Computing* 6 (2): 323–350.

[6] Hume and Sunday (1991) *[Fast String Searching] Software—Practice And Experience*, vol. 21(11), 1221–1248 (November 1991)

[7] Shapiro, Linda; George Stockman (2001). *Computer Vision*. Upper Saddle River, NJ: Prentice Hall.

STATEMENT

I hereby stated that this paper is copyrighted to myself, neither a copy from other‘s paper nor a translation of similar paper.

Bandung, 18th of May 2014

Annisaur Rosi Lutfiana – 13512088