Measuring memetic algorithm performance on image fingerprints dataset

Priati Assiroj1, H. L. H. S. Warnars2, E. Abdurrahman3, A. I. Kistijantoro4, A. Doucet5
1,2,3Computer Science Department Binas Graduate Program, Bina Nusantara University Jakarta, Indonesia
1Information System Department, Universitas Buana Perjuangan Karawang, Indonesia
1Politeknik Imigrasi, Ministry of Law and Human Right, Republic of Indonesia, Indonesia
4School of Electronics Engineering and Informatics, Institut Teknologi Bandung, Indonesia
5L3I Laboratory, La Rochelle University, France

ABSTRACT

Personal identification has become one of the most important terms in our society regarding access control, crime and forensic identification, banking and also computer system. The fingerprint is the most used biometric feature caused by its unique, universality and stability. The fingerprint is widely used as a security feature for forensic recognition, building access, automatic teller machine (ATM) authentication or payment. Fingerprint recognition could be grouped in two various forms, verification and identification. Verification compares one on one fingerprint data. Identification is matching input fingerprint with data that saved in the database. In this paper, we measure the performance of the memetic algorithm to process the image fingerprints dataset. Before we run this algorithm, we divide our fingerprints into four groups according to its characteristics and make 15 specimens of data, do four partial tests and at the last of work we measure all computation time.

Keywords:
Biometrics
Fingerprints
Image
Memetic algorithm
Performance

This is an open access article under the CC BY-SA license.

1. INTRODUCTION

The increase of digital crime, such as data manipulation and image and signature falsification, and many various illegal transactions, is equal to the request of its solutions. Computer-based biometrics validations are important tools to improve the system's security. The most important of biometrics are measure physiology and behavior characteristics that allow making authentication to personal identity. Biometric authentication in the computer-based application is an important thing caused by the huge of sensitive data in the computer system and it increases rapidly.

Personal identification has become one of the most important terms in our society regarding access control, crime and forensic identification, banking, and also computer system [1]. Biometric features that can be used to identification include the iris, voice, DNA, and fingerprints. According to [2], the fingerprint is the most used biometric feature caused by its unique, universality, and stability. The fingerprint is widely used as a security feature for forensic recognition, building access, automatic teller machine (ATM) authentication, or payment. Automatic fingerprint identification has become an interesting research topic for two decades [3], in accordance with [4] fingerprint because of its unique, size, and peculiarity. Nowadays, to get a fingerprint

Journal homepage: http://journal.uad.ac.id/index.php/TELKOMNIKA
recognition tool is very easy, many corporations and organizations use it to identify their member [1]. Fingerprint recognition could be grouped into two various forms, verification [5] and identification [6]. Verification compares one on one fingerprint data. Identification is matching input fingerprint with data that saved in the database. Therefore, identification is an extension of verification that compare one fingerprint data to many, and note that identification is more complex than verification.

2. RESEARCH METHOD

2.1. Memetic algorithm

Memetic algorithm (MA) is an algorithm based on the Neo-Darwinian Evolution concept and Dawkins' opinion about memes as cultural evolution unit which can make improvements to itself. MA is a heuristic method which has similar characteristic to genetic algorithm (GA), combined with local search method, that can improve quality solutions [7]. On MA, local search is used for local improvement which can be applied before or after selection, crossover, and mutation process. Local search is also very useful to control search space. MA can achieve a better result than GA but need more computational time.

MA is a simple algorithm but flexible and powerful [8, 9], that can find quality solutions in many challenging problems [10-12]. Optimization problems will involve dozens of variables thus need good coding devices to overcome computational time. This algorithm will very useful in data mining process such as agglomeration and text analysis [13, 14], biomedical, such as DNA and molecular simulation [15-17], network problems [18, 19], feature selection technique [20], molecular simulation [16], forecasting [21], quantum chemistry [22], spectroscopy analysis [23], geophysics analysis [24], medicine invention [25], genome study [26], and many more.

Specifically, the use of algorithms for image processing has been done by [27], they have used memetic algorithm for image brightness enhancement; [28] use this algorithm to detect face emotion, while [29] use this algorithm for letter recognition. Handwritten recognition conducted by [30] and [31], retina recognition by [32], sub-pixel mapping imagery by [33], image classification by [34], and fingerprint matching with the memetic algorithm by [35], [36] and [5].

2.2. Biometrics

Biometrics are special characteristics of human which is unique on every individual and can be a reference of identification and verification. Biometrics characteristics can be categorized into two types, physical and behavior. We all know that physical types such as fingerprint, iris, retina, and face, also behavior type such as voice, signature, gait, palm geometry, handwritten, electrocardiograph (ECG) [35]. The fingerprint is one of the most reliable biometrics features caused by its unique, and it is impossible to find the same fingerprint between two different people. This biometrics feature can automatically recognize someone by pattern recognition and determine authenticity physiology characteristics.

2.2. Fingerprints

The fingerprint is a graphic ridge and valley pattern at the tip of a human finger. Human fingerprints can be found on many historical objects as shown in Figure 1. This finding proves that ancient people were aware and gave special attention to fingerprint individuality, despite they did not have a scientific basis [37]. The history of fingerprint begins in 1684, morphologist from England, Nehemiah Grew, published his scientific paper about the ridge, groove, and pore structure of fingerprint [37]. In 1788 Mayer has described fingerprint in detail [38].

Figure 1. Fingerprints on historical objects, (a) neolithic sculpture [38], (b) stonehenge at Goat Island [37], (c) clay cup from China 300 BC [37], (d) traces on the light of Palestinian 400 AD [38]
Thomas Bewick in 1809 uses the fingerprint as its characters and it is believed as the milestone of scientific fingerprint [38]. In 1823, Purkinje classifies fingerprint as the first. He was classified fingerprint to nine categories based on the ridge configuration, and in 1880 Henry Fauld and Herschel have done scientific fingerprint recognition [38]. This invention is the basis of modern fingerprint recognition. At the end of the nineteenth century, Sir Francis Galton researched more about fingerprints. Galton introduces a small feature for fingerprint matching in 1888. The most important progress in fingerprint recognition occurred in 1899 when Edward Henry generates "Henry System" [37]. So at the beginning of the twentieth century, the form of fingerprints can be understood well.

3. IMPLEMENTATION MODEL

There are several processes conducted to implement this algorithm. We start the process with reading folders and fingerprint files, we will do the local search process. Then convert fingerprint files to string array form and convert string array to binary code form with Base64, the process will be looped until the entire data converted successfully. The next process is selection or elitism for parent candidates from the total population then make it crossing-over to get new offspring. The last process is a mutation or clones the offspring. Figure 2 shows the flowchart of the process of the memetic algorithm, and a detailed process will be described in the next paragraph. In this paper we implement MA on 7200 synthetic image fingerprint dataset of FVC2006 with characteristics as shown in Table 1.

![Flowchart of memetic algorithm](image)

Figure 2. Flowchart of memetic algorithm

<table>
<thead>
<tr>
<th>Groups</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>100% fully-sized fingerprints image</td>
</tr>
<tr>
<td>B</td>
<td>60% fingerprints with dark color boundaries</td>
</tr>
<tr>
<td>C</td>
<td>60% fingerprints with bright color boundaries</td>
</tr>
<tr>
<td>D</td>
<td>80% fingerprints with bright color boundaries and unclear image</td>
</tr>
</tbody>
</table>
Then we conduct several steps as described below:

- **Local search.** We start the local search by reading the folder and fingerprint file identification and data processing. We divide image fingerprint data into four groups, as shown in Table 1, based on its characteristics.

- **Conversion.** In this step, we convert fingerprints image data to string array then convert string array to binary code using Base64. This algorithm will do some swap, crossover, and mutation process, inside there, are crossing and data cloning that needs binary input and output, so we need it to be converted to binary code. We present the code in java language to convert string array to binary code as shown in Figure 3.

- **Elitism.** In this step, we conduct some selection steps. Theoretically, in elitism, we need to retain 20% random sampling of the population. In the preliminary experiment, we only use a 2% random sampling of the population caused by the limitation of our hard drive. Figure 4 shows the elitism code with java language.

- **Crossover.** Crossover is a hybridization step of every selected sample to get new offspring. The selected samples divide into two groups, male and female, then they will be swapped one to the whole selected sample in another group, this process will be looped until the last selected sample. Figure 5 is the illustration of the crossover process in this paper and Figure 6 shows crossover code in java language.

- **Mutation.** This is the last process, where new offspring from the crossover process cloned then reversed several binary lines to get new offspring. Figure 7 shows the code of the mutation process in java language.

All steps were coded in java language with Netbeans IDE 8.2 and Experiments carried out use host and guest operating system in 1 PC with VMBox. This system runs in Intel i5-2540M with 16 GB of memory.

![Figure 3. Read file or folder and conversion code in Java](image-url)

This work consists of several steps. At the first step we conduct some steps as describe below:

- Read fingerprints image file in every folder and we read one by one.
- Convert image data to array string type using Base64.
- Convert array string to binary code. This is a must because we use binary type in the swap or crossover process and mutation process.
- Population filtering to select parent candidates. In the first experiment, we use 20% of the population then we got swap or crossover result was too big. Caused by the limitation of our hard drive, then we decided to use only 2% of the population so that the swap results will not overwhelm the hard drive.
- Swap or crossover process, mating all of the 2% of the population to get the new offspring. The process is to mate every binary to each other then we got 4900 new offsprings.
- The last process is the mutation for 4900 new offsprings from the crossover process. This is a binary reverse process where each value of 1 will be reversed to 0 and 0 will be reversed to 1.

These codes above are to read every folder that contains fingerprints image data, then converted to array string type using the Base64 process and the result then converted to binary code because crossover and mutation process is mate and clone process of binary code. Figure 4 is the elitism code. Select randomly from 20% of the population to find parent candidates and because of the limitation of our hard drive we use only 2% of the population. The last “null” code is to empty the variable when it is not used so that RAM is not full.

Figure 5 is a crossover function code to mate every selected sample to get the new offsprings where the selected sample divided into 2 groups, male and female then swap to all samples to the other group and looped it. The swap process is depicted in Figure 6 starting from sample 1 in one group mated with all samples in the other group. Likewise with sample 2 and others. This process will be repeated continuously until all the samples have been successfully mated. The last process is the process of mutation. The program code is in Figure 7. Where the results of the marriage from the crossover process are cloned to then reverse the binary code, the value of 1 becomes 0 and vice versa, to get new offspring.

```java
public void getElitism(int randomSampling) {
    List<String> nameFolders = dic.getNameFolders();
    for (String nameFolder : nameFolders) {
        int hitungData = dic.getCountByArrayFingerprint(nameFolder);
        double limitData = hitungData * (randomSampling * DoubleValue() / 100);
        List<String> byteArrays = dic.getByArrayFingerprintRandom(limitData, intValue(), 0);
        for (String byteArray : byteArrays) {
            DataElitismPre elitPre = new DataElitismPre();
            elitPre.setFolder(nameFolder);
            String[] dataPre = byteArr.split(”-”);
            elitPre.setValue(Integer.valueOf(dataPre[0]));
            elitPre.setIsByteFile(dataPre[1]);
            dic.inputDataPreElitism(elitPre);
            dataPre = null;
            elitPre = null;
            byteArrays = null;
        }
    }
}
```

Figure 4. Elitism code in Java

```java
public void getCrossover() {
    int hitungData = dic.getCountElitism();
    Integer limit = hitungData / 2;
    List<DataElitismPre> elitList = dic.getDataByteArrayAfterElitism(limit, 0);
    List<DataElitismPre> elitListInner = dic.getDataByteArrayAfterElitism(limit, (limit * 2));
    for (DataElitismPre elit : elitList) {
        String male = elit.getStringByElitism();
        for (DataElitismPre elitInner : elitListInner) {
            long now = System.currentTimeMillis();
            StringBuilder sbBuild = new StringBuilder();
            String married = sbBuild.append(male).append(elitInner.getValueString()).toString();
            int keyFemale = elitInner.getKey();
            DataContainer innerData = new DataContainer();
            innerData.setByteFile(married);
            innerData.setByteFile(married);
            innerData.setKey(male, innerData.getKey());
            innerData.setKey(male, innerData.getKey());
            innerData.setKey(male, innerData.getKey());
            innerData.setKey(male, innerData.getKey());
            innerData.setKey(male, innerData.getKey());
            dic.inputDataCrossOver(innerData);
            elitListInner = null;
            elitList = null;
        }
    }
}
```

Figure 5. Crossover code in Java language
4. EXPERIMENTAL RESULT

According to Table 2, experiments carried out with 15 specimens of data. Once specimen consists of fingerprint data that grouped before. The following is a description of each specimen:

Specimen 1 consists of the entire fingerprint data
Specimen 2 consists of fingerprint data type A
Specimen 3 consists of fingerprint data type B
Specimen 4 consists of fingerprint data type C
Specimen 5 consists of fingerprint data type D
Specimen 6 consists of a combination of fingerprint data type A to type B
Specimen 7 consists of a combination of fingerprint data type A to type C
Specimen 8 consists of a combination of fingerprint data type A to type D
Specimen 9 consists of a combination of fingerprint data type B to type C
Specimen 10 consists of a combination of fingerprint data type B to type D
Specimen 11 consists of a combination of fingerprint data type C to type D
Specimen 12 consists of a combination of fingerprint data type A, B, and C
Specimen 13 consists of a combination of fingerprint data type A, B, and D
Specimen 14 consists of a combination of fingerprint data type B, C, and D
Specimen 15 consists of a combination of fingerprint data type A, C, and D

According to these specimens, we conduct four steps of experiment, consists of:
- Partial Test 1. In this part, we start with reading fingerprint image data, convert data to array string, and convert array string to binary code.
- Partial Test 2. This is a selection process which uses elitism principal. We select parent candidates 2% of the entire population.
- Partial Test 3. This part is a swap or crossover process to generate new offspring.
- Partial Test 4. This is the last where we do mutation to each offspring.

We measure every processing time in millisecond for each partial test then add them up. Following the results test specimen, 2 achieved the best result and has the most efficient processing time with 109168 ms. The specimen 1 is the worst compared to the entire specimen with total processing time 4032220 ms. This algorithm has processed every specimen successfully with an average of total processing time 1516986 ms.

Table 2. Numerical experiment result

<table>
<thead>
<tr>
<th>Specimen</th>
<th>Partial Test (ms)</th>
<th>Total Time (ms)</th>
<th>Result Data</th>
<th>Data Storage</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1972509</td>
<td>787872</td>
<td>Original 4900</td>
<td>22.8 GB</td>
</tr>
<tr>
<td>2</td>
<td>217672</td>
<td>17498</td>
<td>140</td>
<td>270 MB</td>
</tr>
<tr>
<td>3</td>
<td>212153</td>
<td>63946</td>
<td>35</td>
<td>4.8 GB</td>
</tr>
<tr>
<td>4</td>
<td>215656</td>
<td>70968</td>
<td>289</td>
<td>4.3 GB</td>
</tr>
<tr>
<td>5</td>
<td>127978</td>
<td>44162</td>
<td>289</td>
<td>2.4 GB</td>
</tr>
<tr>
<td>6</td>
<td>472539</td>
<td>339992</td>
<td>289</td>
<td>8.9 GB</td>
</tr>
<tr>
<td>7</td>
<td>363733</td>
<td>279576</td>
<td>289</td>
<td>7.9 GB</td>
</tr>
<tr>
<td>8</td>
<td>261118</td>
<td>212052</td>
<td>289</td>
<td>4.5 GB</td>
</tr>
<tr>
<td>9</td>
<td>745542</td>
<td>245119</td>
<td>289</td>
<td>11.4 GB</td>
</tr>
<tr>
<td>10</td>
<td>646666</td>
<td>176031</td>
<td>289</td>
<td>8 GB</td>
</tr>
<tr>
<td>11</td>
<td>587601</td>
<td>245119</td>
<td>289</td>
<td>7.6 GB</td>
</tr>
<tr>
<td>12</td>
<td>159725</td>
<td>1225</td>
<td>289</td>
<td>7.6 GB</td>
</tr>
<tr>
<td>13</td>
<td>158062</td>
<td>279576</td>
<td>289</td>
<td>17.4 GB</td>
</tr>
<tr>
<td>14</td>
<td>115808</td>
<td>212052</td>
<td>289</td>
<td>13.1 GB</td>
</tr>
<tr>
<td>15</td>
<td>159725</td>
<td>1225</td>
<td>289</td>
<td>15.8 GB</td>
</tr>
</tbody>
</table>

5. CONCLUSION

In this paper, we divide the image fingerprint data into 15 specimens to do four partial tests. In order to know the performance of MA, we measure every specimen in every partial test then generate each processing time. According to the results, specimen 2 achieves the best processing time with 1019168 ms, it has the most efficient processing time of all specimens. At the feature, we will conduct some experiments to this algorithm performance in many environmental systems, such as in a computer network.

ACKNOWLEDGEMENTS

This work is supported by Research and Technology Transfer Office, Bina Nusantara University as a part of Bina Nusantara University’s International Research Grant entitled MEMETIC ALGORITHM IN HIGH-PERFORMANCE COMPUTATION with contract number: No.026/VR.RTT/IV/2020 and contract date: 6 April 2020.

REFERENCES


Measuring memetic algorithm performance on image fingerprints dataset... (Priati Assiroj)
BIOGRAPHIES OF AUTHORS

Priati Assiroj, was born in Cirebon, Jawa Barat, Indonesia in 1986. She has Bachelor and Master in Computer Science. She received the Bachelor from STMIK Bani Saleh Bekasi, in 2011 and received her Master from STMIK LIKMI, Bandung, Indonesia, in 2016. From 2014 to 2016, she was a lecturer in Universitas Singaperbangsa Karawang, Indonesia and from 2016 and recent she is a lecturer in Universitas Bina Perjuangan Karawang in Information System Dept. Since January 2019 she is a lecturer in Politeknik Imigrasi, Ministry of Law and Human Right, Republic of Indonesia. Since March 2018 she has been a scholar of Bina Nusantara Graduate Program, Doctor of Computer Science, Bina Nusantara University Jakarta, Indonesia. Her research fields are data mining, high performance computing and evolutionary algorithm.

H. L. H. S. Warnars, received a Ph.D. degree in Computer Science from Manchester Metropolitan University. Since September 2015 he is a Head of Information Systems concentration at department Doctor of Computer Science Bina Nusantara University, works some project research with my doctoral computer Science students in research area such as Game, Artificial Intelligence including Data Mining, Machine Learning and Decision Support System application such as DSS, BI, Dashboard, Data Warehouse, and so on.

Edi Abdurrachman, received B. Sc and Master of Statistics in Applied Statistics from Bogor Agricultural University, then received M.Sc and Ph.D. in survey statistics and statistics from IOWA State University, USA. He is currently a professor and dean of the Bina Graduate Program, Doctor of Computer Science, Bina Nusantara University Jakarta. His research interest includes statistics, survey statistics, and applied statistics and management information systems. Mr. Abdurrachman’s awards and honors include the MU SIGMA RHO Society (1985) and Best Lecturer Binus University (2012). He is also a member of the American Statistical Association, International Association of Engineers (IAENG), Gamma Sigma Beta, and as a Vice President of the Asian Federation for Information Technology in Agriculture. From 1980-2015 actives in the ministry of agriculture in many positions of director. He is also active as a public speaker in national and international seminars.

Achmad I. Kistijantoro, received the B.Eng. degree in informatics from the Institute of Technology Bandung, (ITB), Bandung, Indonesia, the masters’ degree from TU Delft, Delft, The Netherlands, and the Ph.D. degree from the University of Newcastle upon Tyne, Newcastle upon Tyne, U.K., His current research interests includes distributed systems, parallel computation, and high-performance computation.

Antoine Doucet is a Full Professor in computer science at the L3i laboratory of the University of La Rochelle since 2014. He leads the research group in document analysis, digital contents and images (about 40 people) and is additionally the director of the ICT department of the Vietnam-France University of Science and Technology of Hanoi. Additionally, he is the principal investigator of the H2020 project NewsEye, running until 2021 and focusing on augmenting access to historical newspapers, across domains and languages. He further leads the effort on semantic enrichment for low-resourced languages in the context of the H2020 project Embedia. His main research interests lie in the fields of information retrieval, natural language processing and (text) data mining. The central focus of his work is on the development of methods that scale to very large document collections and that do not require prior knowledge of the data, hence that are robust to noise (e.g stemming from OCR) and language-independent. Antoine Doucet holds a PhD in computer science from the University in Helsinki (Finland) since 2005, and a French research supervision habilitation (HDR) since 2012.