2009

An Automata Based Text Analysis System

Yue Lu
San Jose State University

Follow this and additional works at: https://scholarworks.sjsu.edu/etd_projects

Part of the Computer Sciences Commons

Recommended Citation
Lu, Yue, "An Automata Based Text Analysis System" (2009). Master's Projects. 70.
DOI: https://doi.org/10.31979/etd.4c3t-d352
https://scholarworks.sjsu.edu/etd_projects/70

This Master's Project is brought to you for free and open access by the Master's Theses and Graduate Research at SJSU ScholarWorks. It has been accepted for inclusion in Master's Projects by an authorized administrator of SJSU ScholarWorks. For more information, please contact scholarworks@sjsu.edu.
AN AUTOMATA BASED TEXT ANALYSIS SYSTEM

A Writing Report

Presented to

The Faculty of the Department of Computer Science

San Jose State University

In Partial Fulfillment

Of the Requirements for the Degree

Master of Science

By

Yue Lu

December 2009
Dedicated to
My parents Hong Lu and Leanne Wang
And my Grandma Shaojie
Acknowledgments

I would like to thank my advisor Tsau Young Lin for his guidance and insight, without which I would never complete my project. He helped me for understanding and working on the project.

Moreover, I would give appreciation to Dr. Robert Chun and Dr. Sin-Min Lee for participating in my committee members.

My special thanks go to Dr. Cay Horstmann. Without his email, I would not know what is needed to do for graduating.

I would like to thank Shangxuan Zhang for her advising on my project.

Finally, I would like to thank Mingfang, Qun, Dongyi, Jindou, Kevin and all other friends and family members who have supported me. I thank my parents, Hong Lu and Leanne Wang, for giving me support to explore my interests.

Thank you all!
Abstract

This report describes and implements an automata based text analysis system. We have collected some of the writing samples. Each sample establishes a tree, and uses the ALERGIA algorithm to merge all compatible nodes in order to get a merged stochastic finite automaton. We store these automatons which demonstrate writing style of the sample texts in the hard drive. For a new testing piece, we can test if it has similar writing style compared to those sample texts.

Keywords: Automaton, ALERGIA Algorithm, PTA
# Table of Contents

1. INTRODUCTION ......................................................................................................................... 1  
2. STOCHASTIC FINITE AUTOMATON .............................................................................................. 2  
3. ALERGIA ALGORITHM .................................................................................................................. 3  
4. AUTOMATA BASED TEXT ANALYSIS MODEL .............................................................................. 5  
   4.1 First Example of Automata Based Modeling .............................................................................. 5  
   4.2 Second Example of Automata Based Modeling ........................................................................ 6  
5. IMPLEMENTATION OF METHOD ................................................................................................. 9  
   5.1 Main Classes of Implementation ............................................................................................ 9  
   5.2 Main Functions of Implementation ....................................................................................... 11  
6. SOFTWARE APPLICATION INSTRUCTION .................................................................................. 15  
   6.1 Main Interface of Software .................................................................................................... 15  
   6.2 Setting Parameter ............................................................................................................... 16  
   6.3 Add Sample Text and Analyze It .......................................................................................... 17  
   6.4 Add Test Piece and Test It .................................................................................................. 20  
7. TEST RESULTS ............................................................................................................................ 25  
   7.1 Testing Same Author .......................................................................................................... 25  
   7.2 Testing Different Author .................................................................................................... 26  
   7.3 Twilight vs. Partial Twilight ............................................................................................... 27  
   7.4 Twilight vs. StarWar Episode4 ............................................................................................ 28  
   7.5 Automaton and Match Sequence ........................................................................................ 29  
8. CONCLUSION ............................................................................................................................... 31  
9. FUTURE WORK ............................................................................................................................ 32  
10. REFERENCES .............................................................................................................................. 33  
11. APPENDIX A ............................................................................................................................ 35  
12. APPENDIX B ............................................................................................................................ 37
List of Tables

Table 1. ALERGIA Algorithm Results .............................................................................................. 7
Table 2. Testing Same Author ........................................................................................................... 26
Table 3. Testing Different Author .................................................................................................... 27

List of Figures

Figure 1. Example of DFA .................................................................................................................. 2
Figure 2. First Example of PTA .......................................................................................................... 6
Figure 3. Second Example of PTA .................................................................................................... 7
Figure 4. Example of SFA .................................................................................................................. 8
Figure 5. Classes in the Program ...................................................................................................... 10
Figure 6. Function Words Sequences ............................................................................................... 13
Figure 7. The Main Interface ........................................................................................................... 16
Figure 8. Setting Window .................................................................................................................. 17
Figure 9. Add Analysis File1 ............................................................................................................ 18
Figure 10. Add Analysis File2 ......................................................................................................... 18
Figure 11. Analyze File Complete ................................................................................................... 19
Figure 12. Analyze Result1 .............................................................................................................. 19
Figure 13. Analyze Result2 ............................................................................................................. 20
Figure 14. Add Test File ................................................................................................................... 21
Figure 15. Choose Sample File ........................................................................................................ 22
Figure 16. Test Result for Sample1 .................................................................................................. 23
Figure 17. Test Result for Sample2 ................................................................................................ 24
Figure 18. Test Result for Sample3 ................................................................................................ 24
Figure 19. Test Result for Twilight ................................................................................................ 28
Figure 20. Test Result for Twilight & Starwars4 ........................................................................... 29
Figure 21. Sample5 .......................................................................................................................... 30
Figure 22. Automaton for Sample5 .................................................................................................. 30
Figure 23. Test3 ................................................................................................................................ 31
Figure 24. Matching Sequences ....................................................................................................... 31
1. Introduction

Previously, Tsau Young Lin, Shangxuan Zhang: An Automata Based Authorship Identification System. PAKDD Workshops 2008: 134-142 has illustrated a new method to analyze text base on the automata theory. It has implemented the method to test if an anonymous writing piece has the similar writing style with one sample text from an author in order to verify Authorship Authorization.

Everyone has their own writing characters, depending on his or her gender, age, experience, knowledge, etc. It is been demonstrated through several statistic writing characters, such as word frequency, word length, and sentence length, etc. [10] Given an anonymous writing piece, compared to the sample texts which already learned, we can obtain those writing characters and analyze the texts [1].

The goal of this paper is to continuously study the text analysis method based on the theory of automaton. [10] More precisely, we collect several writing samples. We first get the stop words sequences for each sample text. We then use the ALERGIA algorithm to build a stochastic finite automaton which represents certain writing pattern of the text. Then we analyze those automatons to do future research.

Our program stores patterns for several sample texts. For the anonymous testing piece, we get the sequences of stop words and compare to the stored automatons one by one. We then get the percentages of sentences which accepted by those automatons. The result would be high if the testing piece has the similar writing style compared to the sample text. However, the result might not be such accurate when running multiple testing. We recommend not using this method itself to be only reference.

This paper is structured in seven major sections. In section 2, we first give a description on deterministic finite automata and stochastic finite automaton. In section 3, we describe the ALERGIA algorithm which is used to build an approximated automaton from original sample texts. In section 4, we describe the method which is applied in our program. In section 5, we introduce the implementation and main functions of the software application. In section 6, we give an instruction on how to
use the software. In section 7, we show several results and corresponding automaton of the texts. Finally in section 8 and 9, we conclude the method and explore for possible future work.

2. Stochastic Finite Automaton

In this section, we describe the notion of both deterministic finite automata and stochastic finite automata [2, 3]. Stochastic finite automaton is c.

2.1 Deterministic Finite Automaton

A deterministic finite automaton (DFA) is a 5-tuple \((Q, A, \delta, q_0, F)\), where [3]

- \(Q\) is a finite set of states
- \(A\) is a finite set of input symbols called *alphabet*
- \(\delta: Q \times A \rightarrow Q\) is the transition function
- \(q_0 \in Q\) is a start state
- \(F \subseteq Q\) is a set of accepting states

One simplest DFA is an open/close door sensor which shows in figure 1[3]. The sensor records whether the door is in the "opened” state or the “closed” state.

![Figure 1. Example of DFA](image)

2.2 Stochastic Finite Automaton
A stochastic finite automaton (SFA) is a 4-tuple \((Q, A, q_0, P)\), where

- \(Q\) is a finite set of states
- \(A\) is a finite set of input symbols called *alphabet*
- \(q_0 \in Q\) is an initial state
- \(P\) is a set of probability matrices \(p_{ij}(a)\) giving the probability of a transition from state \(q_i\) to \(q_j\) led by the symbol \(a \in A\). \([2]\]

If we call \(p_{if}\) the probability that the string ends at node \(q_i\), then we have the following constraint \([2]\):

\[
p_{if} + \sum_{q_j \in Q} \sum_{a \in A} p_{ij}(a) = 1.
\]

This means for each state \(q_i\), the probabilities which starts at \(q_i\) plus the probabilities which ends at \(q_i\) equals 1.

### 3. ALERGIA Algorithm

In this section, we describe the ALERGIA algorithm which is used to build an approximated SFA from given set of strings. The original idea comes from R.C.Carraso and J.Oncina: *Learning stochastic regular grammars by means of a state merging method*. Proceedings of the 2nd International Colloquium on Grammatical Inference. Lecture Notes in Artificial Intelligence (1994) 139-152.

The ALERGIA algorithm first builds a prefix tree automaton (PTA) based on the given sample strings. The PTA is a stochastic finite automaton representing all prefixes found in the sample, where each transition is given a probability according to the number of times it is traversed during construction of the PTA \([2]\). Through merging all equivalent and compatible states in the PTA, the algorithm regenerates a SFA. This SFA is an approximation of the original SFA.
Suppose we have set of strings \( S = \{s_1, s_2... s_n\} \). For each string \( s_i = a_1a_2...a_l \), first, we put a start node \( q_0 \). Following the transition \( a_i \), we move to next node \( q_i \) and continue this process until it reaches a node that accepts this string. [10] While running through all strings, we record some statistic data for future usage [2].

Two nodes are said to be equivalent if they have the same outgoing transition probabilities for every symbol \( a \in A \) and the destination nodes must be equivalent also [2]. In symbols, we have

\[
q_i \equiv q_j \Rightarrow \forall a \in A, \text{ we have } p_i(a) = p_j(a) \text{ and } \delta_i(a) = \delta_j(a).
\]

However, we hardly have exactly same frequencies in experimental results. Nodes are accepted to be equivalence within a confidence range. [2]

A confidence range for a Bernoulli variable with probability \( p \) and frequency \( f \) out of \( n \) tries is given by the Hoeffding bound as follows [2]:

\[
\left| p - \frac{f}{n} \right| < \sqrt{\frac{1}{2n} \log \frac{2}{\alpha}} \text{ with probability larger than } (1 - \alpha).
\]

If two estimated probabilities are different in an amount more than the sum of confidence ranges, the ALERGIA algorithm will reject equivalence. [2]

\[
\left| \frac{f}{n} - \frac{f'}{n'} \right| > \sqrt{\frac{1}{2} \log \frac{2}{\alpha}} \left( \frac{1}{\sqrt{n}} + \frac{1}{\sqrt{n'}} \right).
\]

We use above constraint to merge compatible nodes in order to get a SFA which is an approximation of the original one. [10]
4. Automata Based Text Analysis Model

In this section, we describe the text analysis modeling based on automata theory.

4.1 First Example of Automata Based Modeling

We use automata modeling based on stop words. The stop words are predetermined. First, we collect several writing samples. For example, we demonstrate the idea using following paragraph as writing sample. These sentences are cited from *Breaking Dawn* by Stephanie Meyer.

"Childhood is not from birth to a certain age and at a certain age. The child is grown, and puts away childish things. Childhood is the kingdom where nobody dies."

In above example, we have three sentences. We use one sentence as a sequence unit. Therefore, we have three sequences. For each sequence, we keep the stop words and take out all other words. [10] Following the rule described, we get:

is not from to a and at a  
the is and away 
is the where nobody

We notice that the size of unit was chose would be effect to the result. The bigger the unit is, the nicer result would have and also the longer running time would cost. [10] We classify the stop words to five groups and use number 0, 1, 2, 3, 4 to represent adverb, auxiliary verb, preposition/conjunction, pronoun and number respectively. [10] Following the rule described, we have:

1 0 2 2 3 2 2 3 
3 1 2 0 
1 3 0 3 

Now we start to build the PTA. The PTA shows in figure 2.
Let us take a look at this example. We have three strings in the PTA starting with the node q0. Two strings pass node q1. By previous notation, we have $n_1=2$. Node q1 has two children which are q2 and q13. There is one string from node q1 to q2 follow the transition symbol 0, so $f_1(0)=1$. There is one string from node q1 to q13 follow the transition 3, so $f_1(3)=1$. There is no string ending at node q1, so $f_1(\#)=0$.

In this example, we don’t have sufficient data. Therefore, the approximation is not accurate. When going through a large set of sample strings, the approximation would be very well after merging all compatible nodes. Finally, we can get a merged SFA which represents certain style of the text. And it is used to do the text analysis. [10]

4.2 Second Example of Automata Based Modeling

In order to illustrate merging method of the ALERGIA algorithm, we shall look at the following sample. Suppose we have 12 strings:

{a, b, ab, ba, aba, abab, abab, abab, abab, abab, ababab, ababab},

We can build PTA as figure 3 shows:
From above PTA, we compute the values of \(n_i\), \(f_i(#)\) and \(f_i(A)\) for \(A=a,b\) (0≤\(i\)≤8) in following table.

<table>
<thead>
<tr>
<th>Node i</th>
<th>q0</th>
<th>q1</th>
<th>q2</th>
<th>q3</th>
<th>q4</th>
<th>q5</th>
<th>q6</th>
<th>q7</th>
<th>q8</th>
</tr>
</thead>
<tbody>
<tr>
<td>(n_i)</td>
<td>12</td>
<td>10</td>
<td>2</td>
<td>9</td>
<td>1</td>
<td>8</td>
<td>7</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>(f_i(#))</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>5</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>(f_i(a))</td>
<td>10</td>
<td>0</td>
<td>1</td>
<td>8</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>(f_i(b))</td>
<td>2</td>
<td>9</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>7</td>
<td>0</td>
<td>2</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 1. ALERGIA Algorithm results

Where
\(n_i\) is the number of strings arriving at node \(q_i\) [2].
\(f_i(A)\) is the number of strings following transition \(\delta_i(A)\) [2].
\(f_i(#)\) is the number of strings ending at node \(q_i\) [2].

The quotients \(f_i(a)/n_i\) and \(f_i(#) / n_i\) gives estimate the probabilities \(p_i(a)\) and \(p_i#\) respectively [2].

Through the notation in previous section,

\[
\left| \frac{f}{n} - \frac{f'}{n'} \right| > \sqrt{\frac{1}{2} \log \frac{2}{\alpha} \left( \frac{1}{\sqrt{n}} + \frac{1}{\sqrt{n'}} \right)}. 
\]
ALERGIA algorithm will reject equivalence.

If we let $\alpha = 0.7$, then we have

$$|f_6(#) / n_6 - f_8(#) / n_8| = 0.29 < (1/2 \times \log(2/\alpha))^{0.5} \times ((1/(n_6)^{0.5}) + (1/(n_8)^{0.5})) \approx 0.86$$

$$|f_6(a) / n_6 - f_8(a) / n_8| = 0.29 < (1/2 \times \log(2/\alpha))^{0.5} \times ((1/(n_6)^{0.5}) + (1/(n_8)^{0.5})) \approx 0.86$$

$$|f_6(b) / n_6 - f_8(b) / n_8| = 0 < (1/2 \times \log(2/\alpha))^{0.5} \times ((1/(n_6)^{0.5}) + (1/(n_8)^{0.5})) \approx 0.86$$

It shows that node 6 and node 8 are compatible. No other nodes are qualified to be merged. Therefore, we can merge nodes 6 and node 8 to get the following merged SFA in figure 4.

![Diagram of SFA](image)

**Figure 4. Example of SFA**

Now for another set of strings, we test each string if it can be accepted by this merged SFA. Then we compute the accepting probability for the strings.

For example, we have these 10 strings as our testing data:

{aaab, aaba, ababa, ababab, aaaa, bbba, bb, bab, bbab, baba}, where only one of the strings is same from sample strings. After performing the test, the string “ababab” has been accepted by this SFA. Therefore, the accepting probability equals 1/10 (10%).
Let us take a look at another example. We have following 10 strings: 
{ aaab, aaba, ababa, ababababab, aaaa, bbba, bb, bab, bbaa, baba },
where none of them are same from sample strings. However, after performing the test, the string “ababababab” has been accepted by this SFA. The accepting probability also equals 1/10 (10%), too.

We have realized that the accepting probability depends on the parameter $\alpha$ in our method. [10] If we set $\alpha$ too small ($0 \leq \alpha \leq 1$), it is possible to merge nodes which are not compatible at all. If we set the parameter too large, few states are qualified to merge, the percentage results would be getting low. Usually, we think $\alpha = 0.7$ is a reasonable value to set.

5. Implementation of Method

In this section, we describe major structure of implementation and main functions of the software.

5.1 Main Classes of Implementation

Our program is written in C++. We compile the source code on Windows XP, using Microsoft Visual Studio 2008 as our development environment. Figure 5 shows the class view of the program.
The definition for class *node*:

```cpp
class node
{
public:
    node(void);
    ~node(void);
    long index;
    long parent;
    long num_pass;
    long num_accept;
    bool end;
    bool merge;
    long merge_to;
    long child[5];  // 5 is StopWordType
    long num_appear[5];  // 5 is StopWordType

    friend istream & operator >> (istream &in, node &obj);
    friend ostream & operator << (ostream &out, node &obj);
}
```

Figure 5. Classes in the program
We implement class \textit{node} to store data of the nodes in SFA.

\textit{index} indicates the index of nodes in SFA.

\textit{parent} indicates the parent of the node.

\textit{num\_pass} indicates the number of strings which pass through the node.

\textit{num\_accept} indicates the number of strings which are ended at this node. The value would be zero if the node is not a final state.

\textit{end} indicates if the node is a final state. If it is a final state, the value is set to \textit{true}, otherwise the value is \textit{false}.

\textit{merge} indicates which node need to be merged when applying the ALERGIA Algorithm.

\textit{merge\_to} indicates which node it would merge to when applying the ALERGIA Algorithm

\textit{child[5]} indicates the children of one string, it must have 5 or less different children since we only have 5 types of function words.

\textit{num\_appear[5]} represents the number of strings which pass through by the string for each type of function words.

\textbf{5.2 Main Functions of Implementation}

The definition for class \textit{Analyze}:

\begin{verbatim}
class Analyze :
    public CObject
{
public:
    Analyze(void);
~Analyze(void);

};
\end{verbatim}
static const long StateLimit=1000000; //number of state
static const int WordType=5; //number of stop words
static const int M=1; //sentence num
static const int WordLength=100;
static const int WordNumber=100;
static const int Exception1=10;
static const int Exception2=20;
static const int Exception3=30;

enum {Adv,Aux,Prep,Pron,Number};

public:
  long max_state, trCounter;
  node state[StateLimit];
  long temp[StateLimit];
  long treeEnd[StateLimit];
  long count;
  double progress;
  double a;

public:
  long GetStopWord(CString dir, CString in, CString out_dir, CString out);
  int BuildPTA(CString dir, CString in);
  int Compatible(long node_i, long node_j);
  int Differ(double n_1, double n_2, double f_1, double f_2);
  long Delta(long i, int t);
  int Combine(void);
  int Merge(CString dir);
  int TextAnalysis(CString dir, CString name);
  void Output(CString strFile);
  void Input(CString strFile);
};

We implement class *Analyze* to apply the main method which described in previous sections.

The major functions in class *Analyze* are illustrated as following:

  long GetStopWord(CString dir, CString in, CString out_dir, CString out);
This function is used to get all function words sequences from the sample text. It reads input file word by word, and only keeps those predetermined function words. We define -1 to represent the end of one sentence and 0, 1, 2, 3, 4 to represent five types of stop words respectively. [10] We can obtain sequence as following:

![Function Words Sequences](image)

As meantime, we record some statistic data, such as total number of words and number of function words, etc.

```c
int BuildPTA(CString dir, CString in);
```

This function is used to construct PTA which looks like the figure 2 shows previously. As we get the sequence of numbers from last
function, we start to construct the states of PTA one by one. The result was written in a file named *pta.txt*. Array *state[StateLimit]* stores the nodes of the PTA.

```c
int Compatible(long node_i, long node_j);
int Differ(double n_1, double n_2, double f_1, double f_2);
long Delta(long i, int t);
```

These three functions are used to calculate the statistic data of the PTA. The results are prepared for merging equivalent states in next step.

```c
int Combine(void);
int Merge(CString dir);
```

These two functions are used to merge all equivalent states in PTA. The function *Combine(void)* determines what nodes are needed to be merged, and the function *Merge(CString dir)* does the merging process which updates the children and parents of the nodes. The results are written to a file named *automaton.txt*. This is the final SFA we get which approximate the original SFA to represent certain writing style of the sample text.

```c
int TextAnalysis(CString dir, CString name);
```

This function is used to compare a new writing piece to the sample texts. We get the sequence of number from the testing piece and test each sequence whether it can be accepted by the SFA. [18] Then we compute the accepting probability for the strings.

The definition for class *AnalysisResult*:

```c
class AnalysisResult : public CObject
{
  public:
```
We implement class `AnalysisResult` to manage the results of processing sample data. It creates a folder `AnalysisResult`. In this folder, it creates a file called `AnalysisResult.txt` to store the directory of the sample data. For each sample data, it also creates a folder which is named as the sample title and store the related information about the corresponding sample.

The main idea of class `AnalysisResult` is to save analysis results of each sample for later use. For example, we store three pieces of writing samples, we can test an anonymous writing piece with these three results to see how similar with each of the three samples.

6. Software Application Instruction

In this section we represent how to use the software and the functions of the software application `TM`. The program can be run in Windows XP operating system.

6.1 Main Interface of Software

After open `TM`, we shall see a dialog-based window as figure 6 shows:
In order to run the program, first we need to set the confidence level which is used in the program as a parameter. To do this, we simply click *Setting* button on the right hand side of the window.

### 6.2 Setting Parameter

After clicking *Setting* button, a pop-up window will show up as following figure:
The confidence level has a default value which set to be 0.7. It controls the accuracy of merging nodes. The value should be a number between 0 and 1. If it has been set too small, it is possible to merge nodes which are not equivalent at all. If the value has been set close to 1, very few states can be merged. Usually, we think $\alpha = 0.7$ is an appropriate value.

After setting the confidence level, click OK to go back to the main interface. Now we need to add one sample text and analyze it.

### 6.3 Add Sample Text and Analyze it

After clicking Add Analysis button, a pop-up window will show up as following figure:
From this window, we need to browse to select a sample text file. The results showing as following figure:

Now it is ready for analyzing the data, click *Analysis File* button on the left corner, and wait until the label shows *Analyzing Complete!* as below:
When analyzing complete, a SFA represented the certain writing style of the sample text has been generated. Click OK to go back to main interface. Now we shall see the sample text name already in the analysis result list as below:

Following same steps, we can add several sample texts and analyze them. The results will be show up in the main interface. Following figure shows 3 sample texts have been analyzed:
6.4 Add Test Piece and Test it

Now we can test an anonymous writing piece with the sample data in the result list. First of all, we need to add a file which is going to be tested. Simply click Add Test button and select a file. Then the test file name will show up at the right corner below the label Test File: and the information bar will show similar message said add test file OK which is showing as below:
Now we start to test the file. There are three sample texts which have been stored in the system. We need to select one to be tested. Simply click a place which is in same line as the sample text and text color would be changed which shows as below:

Figure 14. Add Test File
Last step, simply click *Testing* button on the right hand side to let program run. It might take several seconds to process. After it is done, we will see the probability result shown on the information bar as following:
This result means that the test file *Test1.txt* has 32 percent similar writing style as *Sample1.txt*.

You can test it with Sample2 and Sample3 using same steps. Select one and click *Testing* button, the results will show:
Figure 17. Test Result for Sample2

Figure 18. Test Result for Sample3
The two results show that the test file has 14 percent similar writing style when compared to Sample2, and 69 percent similar writing style when compared to Sample3.

From these three different test results, we can say that Test1.txt has more writing style which is more similar to Sample3.txt. Actually they are from same author.

We can add several sample texts, and test each of them with the test file. Or we can change the test file also. Through running our program, we can analyze texts based on function words and automata modeling. The higher the percentage is, the more likely the testing piece has a similar writing style with this sample text.

7. Test Results

In this section we show couple of results, the merged SFA and accepting sequences by running our program. Notice that all confidence value are set to be the same ($\alpha = 0.7$) for these testing.

7.1 Testing Same Author

Sample file and Test file are both from the author Stephanie Meyer ($\alpha = 0.7$)

<table>
<thead>
<tr>
<th>Sample</th>
<th>Test</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>Breaking Dawn by Stephanie Meyer</td>
<td>Eclipse by Stephanie Meyer</td>
<td>69%</td>
</tr>
<tr>
<td>(full version)</td>
<td>(full version)</td>
<td></td>
</tr>
<tr>
<td>Breaking Dawn by Stephanie Meyer</td>
<td>Eclipse by Stephanie Meyer</td>
<td>68%</td>
</tr>
<tr>
<td>(full version)</td>
<td>(half version)</td>
<td></td>
</tr>
<tr>
<td>Breaking Dawn by Stephanie Meyer</td>
<td>Eclipse by Stephanie Meyer</td>
<td>68%</td>
</tr>
<tr>
<td>(full version)</td>
<td>(quarter version)</td>
<td></td>
</tr>
<tr>
<td>Breaking Dawn by Stephanie Meyer</td>
<td>Eclipse by Stephanie Meyer</td>
<td>67%</td>
</tr>
<tr>
<td>(full version)</td>
<td>(1/8 version)</td>
<td></td>
</tr>
<tr>
<td>Breaking Dawn by Stephanie Meyer</td>
<td>Eclipse by Stephanie Meyer</td>
<td>66%</td>
</tr>
<tr>
<td>(half version)</td>
<td>(full version)</td>
<td></td>
</tr>
<tr>
<td>Breaking Dawn by Stephanie Meyer</td>
<td>Eclipse by Stephanie Meyer</td>
<td>64%</td>
</tr>
<tr>
<td>(half version)</td>
<td>(half version)</td>
<td></td>
</tr>
</tbody>
</table>
Breaking Dawn by Stephanie Meyer (half version)  |  Eclipse by Stephanie Meyer (quarter version)  |  64%
Breaking Dawn by Stephanie Meyer (half version)  |  Eclipse by Stephanie Meyer (1/8 version)  |  64%
Breaking Dawn by Stephanie Meyer (quarter version)  |  Eclipse by Stephanie Meyer (full version)  |  61%
Breaking Dawn by Stephanie Meyer (quarter version)  |  Eclipse by Stephanie Meyer (half version)  |  59%
Breaking Dawn by Stephanie Meyer (quarter version)  |  Eclipse by Stephanie Meyer (quarter version)  |  59%
Breaking Dawn by Stephanie Meyer (quarter version)  |  Eclipse by Stephanie Meyer (1/8 version)  |  58%
Breaking Dawn by Stephanie Meyer (1/8 version)  |  Eclipse by Stephanie Meyer (full version)  |  55%
Breaking Dawn by Stephanie Meyer (1/8 version)  |  Eclipse by Stephanie Meyer (half version)  |  53%
Breaking Dawn by Stephanie Meyer (1/8 version)  |  Eclipse by Stephanie Meyer (quarter version)  |  53%
Breaking Dawn by Stephanie Meyer (1/8 version)  |  Eclipse by Stephanie Meyer (1/8 version)  |  52%

Table 2. Testing Same Author

The percentage is getting low when the sample file and test file cut small. However, the result shows test file still has more than half of the sequences have been accepted by the generated SFA from sample file.

### 7.2 Testing Different Author

Sample file and Test file are from different author (α =0.7)

<table>
<thead>
<tr>
<th>Sample</th>
<th>Test</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>Term paper from Yue Lu (full version)</td>
<td>Eclipse by Stephanie Meyer (full version)</td>
<td>14%</td>
</tr>
<tr>
<td>Term paper from Yue Lu (full version)</td>
<td>Eclipse by Stephanie Meyer (half version)</td>
<td>14%</td>
</tr>
<tr>
<td>Term paper from Yue Lu (full version)</td>
<td>Eclipse by Stephanie Meyer (quarter version)</td>
<td>14%</td>
</tr>
<tr>
<td>Term paper from Yue Lu (full version)</td>
<td>Eclipse by Stephanie Meyer (1/8 version)</td>
<td>12%</td>
</tr>
</tbody>
</table>
Table 3. Testing Different Author

<table>
<thead>
<tr>
<th>Term paper from Yue Lu (version)</th>
<th>Eclipse by Stephanie Meyer (version)</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Term paper from Yue Lu (half version)</td>
<td>Eclipse by Stephanie Meyer (full version)</td>
<td>14%</td>
</tr>
<tr>
<td>Term paper from Yue Lu (half version)</td>
<td>Eclipse by Stephanie Meyer (half version)</td>
<td>14%</td>
</tr>
<tr>
<td>Term paper from Yue Lu (half version)</td>
<td>Eclipse by Stephanie Meyer (quarter version)</td>
<td>14%</td>
</tr>
<tr>
<td>Term paper from Yue Lu (half version)</td>
<td>Eclipse by Stephanie Meyer (1/8 version)</td>
<td>12%</td>
</tr>
<tr>
<td>Term paper from Yue Lu (quarter version)</td>
<td>Eclipse by Stephanie Meyer (full version)</td>
<td>14%</td>
</tr>
<tr>
<td>Term paper from Yue Lu (quarter version)</td>
<td>Eclipse by Stephanie Meyer (half version)</td>
<td>14%</td>
</tr>
<tr>
<td>Term paper from Yue Lu (quarter version)</td>
<td>Eclipse by Stephanie Meyer (quarter version)</td>
<td>14%</td>
</tr>
<tr>
<td>Term paper from Yue Lu (quarter version)</td>
<td>Eclipse by Stephanie Meyer (1/8 version)</td>
<td>12%</td>
</tr>
<tr>
<td>Term paper from Yue Lu (1/8 version)</td>
<td>Eclipse by Stephanie Meyer (full version)</td>
<td>14%</td>
</tr>
<tr>
<td>Term paper from Yue Lu (1/8 version)</td>
<td>Eclipse by Stephanie Meyer (half version)</td>
<td>14%</td>
</tr>
<tr>
<td>Term paper from Yue Lu (1/8 version)</td>
<td>Eclipse by Stephanie Meyer (quarter version)</td>
<td>14%</td>
</tr>
<tr>
<td>Term paper from Yue Lu (1/8 version)</td>
<td>Eclipse by Stephanie Meyer (1/8 version)</td>
<td>12%</td>
</tr>
</tbody>
</table>

The percentage is getting low when the sample file and test file cut small. The result is getting nicer to verify different author.

7.3 Twilight vs. Partial Twilight

The following section analyzes text from the book *Twilight* by Stephanie Meyer:

Sample text: Twilight from Stephanie Meyer

Test file: First Twelve Chapters of Twilight from Stephanie Meyer
From this example, *First Twelve Chapters of Twilight* has 99 percent similar writing style compared to the Full version *Twilight*. It means almost all of sequences of patterns have been accepted by the SFA which is generated in analyzing file step. Since they are same book, just cut some text from full version, they should have same writing style, the result convinces it.

### 7.4 Twilight vs. StarWar Episode4

We choose two different author’s book to test:

Sample text: Twilight by Stephanie Meyer

Test file: Starwars Episode 4 A New Hope by Alan Dean Foster
From this example, *Starwars Episode 4 A New Hope* has 54 percent similar writing style compared to *Twilight*. This means only half of sequences of patterns have been accepted by the SFA which is generated in analyzing file step.

We notice that if the parameter $\alpha$ is relatively small, then it would merge a lot of non-equivalent states. The percentage results would be getting high. However, if the parameter is relatively large, few states are qualified to merge, the percentage results would be getting low. It is important to pick up the proper value for the parameter in order to get a better result. [10]

### 7.5 Automaton and Match Sequence

We provide a small case to show the generated automaton and the matching sequence.

Suppose we have *Sample5.txt*:
We generate the SFA as following:

Index means the nodes in the SFA. Since we have 5 types of function words, each node has maximum 5 children. "-" means there is no children from this transaction. For example, node 5 has one child node
6 led by 2 (which is one function word type). And node 6 has no child at all. Therefore, node 6 is a final state.

We have a *Text3.txt*:

![Figure 23. Test3](image)

After testing, we got following accepting probability:

```
  test1 6
test1 11
test1 17
success=3
total=10
prob=0.3
```

![Figure 24. Matching Sequences](image)

From the above figure, we can see that node 6, 11, and 17 has been accepted by the SFA. When we look at the automaton for Sample5.txt, we can see that node 6, 11, and 17 are final state. That means there is no child of these nodes. The three sequences which end at node 6, 11, and 17 are accepted by the SFA. The accepting probability is 0.3.

### 8. Conclusion

We have shown the text analysis method which use automata modeling and the ALERGIA Algorithm. We then showed how it is implemented in our program. Based on the previous result, we have
improved the implementation so that we can store a lot of sample patterns of sample texts. And we let one testing piece to test the probability of similar writing style compared to each of the sample text. It would be better to combine the result with other methods or tools instead of itself [10].

9. Future Work

It will be very interesting and challenging to work on the program following the algorithm. One can use different set of function words, or one can use one paragraph to be a sequence instead of one sentence to refine the result.

Improvements may be achieved with taking out those common sequences that might appear in many texts. Based on our result, some texts from different author still have high similar writing style. This might happen because we didn’t take out those common sequences which are used by anyone. If we could find those common sequences and remove them from the merged SFA, we may get an even sharp accepting probability. The result would be much nicer.

For future thinking, the same method can also be applied to Microarray in bioinformatics to deal with DNA sequence or sequences on Turning Machine. It is an interesting topic to work on and generalize this method combined with other tools.
10. References


Discovery, A. Skowron and L. Polkowski (eds), Physica-Verlag, 1998, 121-140


11. Appendix A

The function words are predetermined. We got the lists from internet and Tsau Young Lin, Shangxuan Zhang: An Automata Based Authorship Identification System. PAKDD Workshops 2008: 134-142.

To store the data into our program, we define the following array

```c
static const char funword[WordType][WordNumber][WordLength]={
    {"absolutely","again","ago","almost","alone","already","also","always","anywhere","away","back","barely","carefully","downtown","else","every","everywhere","far","fast","frequently","hard","hardly","hence","here","hither","home","how","however","immediately","lately","later","mostly","near","nearby","nearly","never","not","now","nowhere","occasionally","often","only","out","pretty","quickly","quite","rarely","rather","really","recently","seldom","slowly","sometimes","somewhere","soon","still","then","thence","there","therefore","thither","thus","today","together","tomorrow","tonight","too","underneath","usually","very","well","when","whence","where","whither","why","yes","yesterday","yet"},
    {"d","ll","s","am","ain't","are","aren't","be","been","being","can","can't","could","couldn't","did","didn't","do","does","doesn't","doing","done","don't","get","gets","getting","got","had","hadn't","has","hasn't","haven't","having","he'd","he'll","he's","i'd","i'll","i'm","is","i've","isn't","it's","may","mayn't","might","must","mustn't","ought","oughtn't","re","shall","shan't","she'd","she'll","she's","should","shouldn't","that's","they'd","they'll","they're","was","wasn't","we'd","we'll","were","we're","weren't","we've","will","won't","would","wouldn't","you'd","you'll","you're"},
    {"aboard","about","above","across","after","against","along","alongside","although","amid","amidst","among","amongst","and","around","as","aside","astride","at","before","behind","below","beneath","beside","besides","between","beyond","but","by","concerning","despite","down","during","except","excluding","following","for","from","given","if","in","including","inside","into","like","minus","near","next","nor","of","off","on","onto","or","out","outside","over","past","per","regarding","round","since","so","than","that","though","through","till","to","toward","towards","trough","under","underneath","until","underway","up","upon","upwards","upward","upside","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside download"},
    {"about","above","across","after","against","along","alongside","although","amid","amidst","among","amongst","as","aside","astride","at","before","behind","below","beneath","beside","besides","between","beyond","but","by","concerning","despite","down","during","except","excluding","following","for","from","given","if","in","including","inside","into","like","minus","near","next","nor","of","off","on","onto","or","out","outside","over","past","per","regarding","round","since","so","than","that","though","through","till","to","toward","towards","trough","under","underneath","until","underway","up","upon","upwards","upward","upside","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside down","upside download"}
};
```
The enumerate type is

```c
enum {Adv,Aux,Prep,Pron,Number};
```

stores the function words we are interested in.
12. Appendix B

The results are obtained by running program on:
Microsoft Window XP Professional Version 2002 Service Pack 3

DELL INSPIRON E1405
Intel CPU CORE DUO T2300 @ 1.66 GHZ
2GB of RAM

Sample1.txt: Study information from internet
Sample2.txt: Term paper from Yue Lu
Sample3.txt: Breaking Dawn by Stephanie Meyer

Test1.txt: Eclipse from Stephanie Meyer

All confidence value are set to be the same ($\alpha = 0.7$) for testing.

The running time is proportional to the number of function words, not proportional to the number of sentences. Usually the longer the text is, the more function words are.

I got different running time for each analyzing and testing process. Mostly it takes less than 15 seconds when the computer doesn’t run other programs at the same time.