GRALT: A MULTI-THREAD TOOL FOR TEXT VERSIONS ANALYSIS

Katia Mayfield
Dept. of Math, Computer and Natural Sciences,
Athens State Univ.,
Athens, AL USA

Ronald Marsh
Department of Computer Science
University of North Dakota,
Grand Forks, ND USA

Crystal Alberts
Department of English
University of North Dakota,
Grand Forks, ND USA

Abstract - *The existence of multiple versions of a text, regardless of how they were created (in print or digital), and the possibility of using a computer as an analytical tool has led to the development of software systems used by textual criticism scholars. Following the tradition of this type of work from initial efforts by IBM to today's Google, this study focuses on the multi-thread implementation of a tool supporting the study of text variations by textual critics.*

Keywords: multi-thread, wavefront, evolution, version, performance

1 INTRODUCTION

Textual critics examine changes caused by editing, authors' modifications, and even printing errors to discover the evolution of text variants. To distinguish between versions, scholars collate texts, which according to Williams and Abbot, is the process where one text is compared with another to discover textual variants [14]. Along with the comparison between texts, a scholar must also take into account that an author's manuscripts may also have notations or marginalia, such as dates, numbers, biographical or historical events, that make the distinction between versions possible. When the texts do not provide a clear idea of such a sequence, then the textual critic may depend on bibliographical studies to provide extra information. Historical, analytical and descriptive bibliographical studies examine published documents, like a book, in order to identify several attributes of those publications.

For example, analytical bibliographic scholars investigate various physical characteristics of a work, including manufacturing processes, ink components, and book binding techniques, to find significant information that will help them with their study. These studies can also be conducted with the goal of trying to establish a chronological sequence of document creation [14]. However, with the advancement of digital technology, new obstacles have surfaced in bibliographical studies. While digital technology leads to greater access to a manuscript's content, its physical characteristics cannot be accessed in the digital realm, at least not in the same way. As such, digital tools for digital objects are necessary. Due to the amount of data to be analyzed and the target user, these tools require high performance solutions to be applicable to office solutions. A parallel implementation based on multithread techniques is therefore the basis of this study.

2 BACKGROUND

A series of recently developed investigative tools, complement Shillingsburg's argument that a new set of characteristics need to be analyzed in documents that are represented in digital form, including hardware (from the processor to the monitor), the communication networks, the character sets, type fonts, mark-up, and formatting of the text [9]. Even blank spaces and printer paper--if the document is printed--become part of this analysis.

Several scholars, Michael Wise among them, have conducted studies on textual variants concentrating in the area of plagiarism [15]. However in the study of plagiarism, the texts are compared in search of segments that are plagiarized. These studies do not consider the possibility of targeting versions of the same work, which is the goal of this research.

Two well-known tools among humanists, the Versioning Machine, built by Susan Schreibman et al, and Juxta, created by the Applied Research in Patacriticism group at the University of Virginia, have been used in a number of projects [3, 12]. Scholars, such as Lara Vetter and Jarom McDonald in "Witnessing Dickinson's Witness," apply the Versioning Machine to demonstrate the complexity of Emily Dickinson's work with respect to the number of published variants [13]. The text is displayed through the Versioning Machine to allow for a comparison between versions of Dickinson's work and to show how Dickinson varied words, phrases, lines and stanzas. Juxta provides the user with histograms, which display the frequency of differences between the initial text to a specific variant, allowing the users to identify sections of the text that have been heavily modified [3]. These tools provide a visualization of the differences between texts, but still require the manual intervention of the textual critic to establish the text evolution, which implies that even though the differences between documents can be seen by the critic, the tool does not provide any information on the evolution of the text.

3 GRALT: A MULTI-THREADED TOOL

This section presents GRALT, Graph Analyzer of Literary Text Versions, a multi-thread solution based on graph techniques, which has a polynomial time complexity. In order for GRALT to estimate the evolutionary sequence of the text versions, a text comparison graph is constructed to represent the differences between the texts [6, 7].

In order to measure the difference between texts, an editing metric technique is required. One of the most widely known text editing metrics is Levenshtein's Distance [5]. It calculates the least number of operations that are required to modify a text, at character level, and obtain a new one. The Levenshtein's algorithm shown in Figure 1, utilizes a dynamic programming technique in a two dimensional matrix to perform the required calculations. The texts being compared are aligned with the rows and columns of the matrix. The matrix is then populated from the upper left corner to the bottom right corner. The number found in the lower right corner is the Levenshtein distance between the documents.

The first approach to finding the evolutionary sequence by using graph traversal algorithms is based on concepts associated with a Hamiltonian path. Such graph nodes represent the text variants while the edges indicate the differences found in the text. In this study, it is assumed that the user has enough information to establish which node corresponds to the original (first draft) of the text, and then a greedy algorithm named Single Path Evolution (SPE), shown in Figure 2, is applied to the graph, adding edges to the solution path in a single direction. The algorithm assumes that the graph is fully connected.

A second approach involves the use of a Minimum Spanning Tree algorithm. Two common algorithms used to find a MST are the Kruskal's algorithm and the Prim's algorithm [8]. A simple pseudo code representation of the Kruskal's algorithm is shown in Figure 3.

```
Levenshtein's Algorithm:
input: text1 and text2
output: Levenshtein's distance
lev[0][j]=j for  0≤j≤ length(text2)
lev[i][0]=i for  0≤i≤ length(text1)
for (i = 1; i ≤ length(text1); i++)
{
  for (j = 1; j ≤ length(text2); j++)
  {
     if( text1 [ i -1] == text2 [ j-1 ] )
     {
          lev[ i ] [ j ] = lev [ i – 1] [ j – 1]
     }
          else
          {
               lev[i ] [ j ] = minimum(lev[i-1][j],
                     lev[i][j-1], lev[i-1][j-1]) + 1
          }
  }
}
return lev[length(text1) ] [ length(text2)]
```

Figure 1 Levenshtein's Algorithm

```
SPE Algorithm
input:  graph G=(V,E, W), original node s
output: shortest Hamiltonian path, H = (M, D)
     D←Ø
     M← {s}
     V ←V - {s}
     end_of_M ← s
     While V ≠ Ø
          select v in V with minimum weight from end_of_M
          D ← D ∪ {(end of M - v)}
          M←M ∪ {v}
          V ←V - {v}
          end_of_M ← v
```

Figure 2 SPE algorithm

```
Kruskal Algorithm
input:  graph G=(V,E, W), original node s
output: Minimum Spanning Tree F=(M,T)
// create a forest (unconnected graph) F such that F=(M,T)
T←Ø
M ←V
sort E into non-decreasing order by weight w
for each (u, v) ∈ E taken from the sorted list
if (u,v) connects 2 different trees in F
then T ← T ∪ {(u, v)}
Return MST F = (V,T)
```

Figure 3 Kruskal's MST algorithm

The combined functionality of a modified Levenshtein algorithm and the version sequence estimation available through the SPE path and the MST algorithms, results in the GRALT algorithm shown in Figure 4. The algorithm produces an undirected graph and considering the assumption that the original is known, it can be transformed into a direct graph, introducing the directions necessary to obtain the estimated version evolution.

4 MULTI-THREAD DESIGN

Significant computer programming challenges, involving memory utilization and computational time, exist in the implementation of GRALT. It is important to implement a multithreaded version of the algorithm with dynamic memory allocation to support the different needs of data storage. The comparison of two texts using the modified Levenshtein's method requires the execution of a nested pair of loops whose total number of iterations may result in a very long execution time. In this study, given the size of some of the problems that are investigated, multi-thread programming based on a technique proposed by Lamport in 1974 to improve the execution of nested loops is used [4].

```
GRALT algorithm
Input: list of texts L, original text s ∈ L
Output: estimated evolutionary sequence of L
// build complete graph G=(V,E,W)
V←L
E←Ø
For all v ∈ V
E ← E ∪ {(v→u)| u ∈(V-{v}})
// compute editing distance between texts
For all $v_i$ ∈V
    For all $v_j$ ∈V, j>i
        W($v_j$,$v_i$) ←Levenshtein($v_i$,$v_j$)
// find the SPE path of G
Estimated sequence ← SPE (G, s)
// or find the minimum spanning tree
Estimated sequence ← Kruskal (G,s)
```

Figure 4 GRALT Algorithm

This technique, known as the wavefront approach, uses a hyperplane concept to identify parallel code in uniform nested loops [4]. In particular, the modified Levenshtein's algorithm has a double nested loop equivalent to a two-dimensional space. Usually, the execution of such loop iterations follows a row-wise or column-wise sequence. In the modified Levenshtein's algorithm, each iteration depends on three values previously calculated: the first value is found in the same column and a previous row, the second on the same row and a previous column and the third in a diagonal, as represented by the arrows seen in Figure 5. The few values that are shown in the matrix are examples of the calculated distance based on the modified Levenshtein's algorithm.

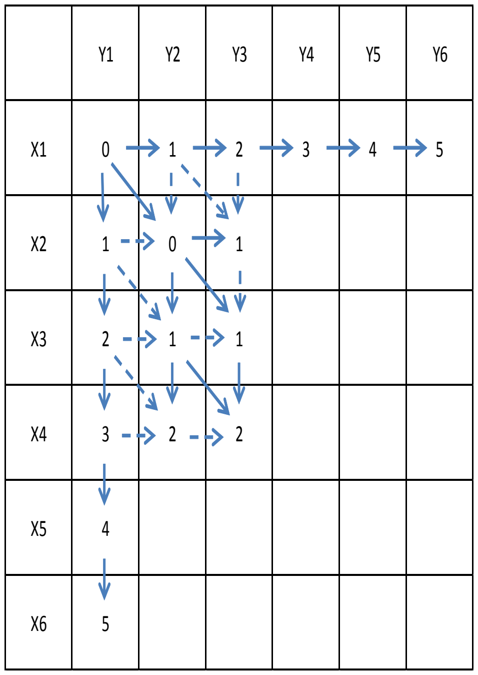
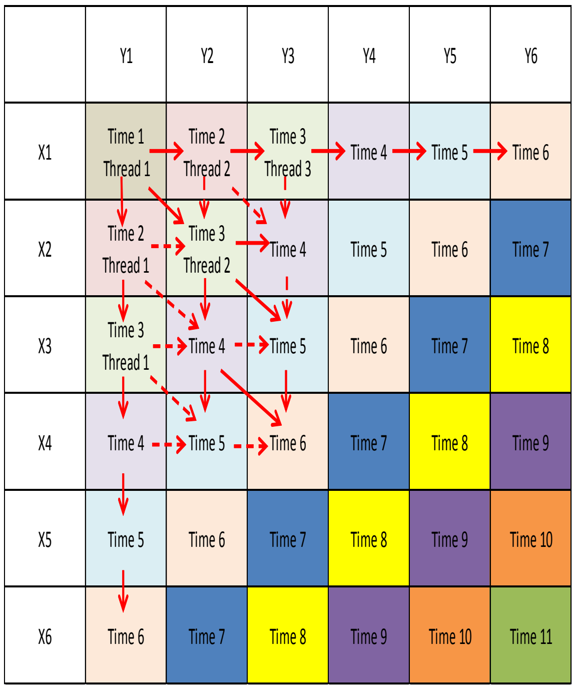


Figure 5 Levenshtein algorithm dependencies

In the Levenshtein's algorithm, it is clear that after iteration (1,1) has been completed, iterations (2,1) and (1,2) could be run in parallel, followed by iterations (3,1), (2,2), and (1,3). In this study, considering the synchronization required by the iterations (to obtain data previously computed), any attempt to parallelize individual iterations might cause undesirable overhead. A better solution is to apply this concept to groups of iterations as represented in Figure 6. A single thread runs the first block of iterations; the second and third blocks are assigned to two threads that will run concurrently, and depending on the number of threads available, the number of concurrent executions will increase. In a simple solution, as seen in Figure 9, if six threads are available, the rows would be split in 6 blocks, and each block of columns would be assigned to one of the threads. Therefore, when getting to the third hyperplane, the group (X3,Y1) would run concurrent with (X2,Y2) and (X1,Y3).

Another computational problem is the amount of memory required by the modified Levenshtein's algorithm. In a simple example, two texts of size

200,000 characters being compared would require a matrix of 200,000 by 200,000 integer values or 40,000,000,000 integer values or approximately 160 Gbytes of memory. Well beyond the typical physical memory available in current desktop computers.



	Y1	Y2	Y3	Y4	Y5	Y6
X1	Time 1 Thread 1	Time 2 Thread 2	Time 3 Thread 3	Time 4	Time 5	Time 6
X2	Time 2 Thread 1	Time 3 Thread 2	Time 4	Time 5	Time 6	Time 7
X3	Time 3 Thread 1	Time 4	Time 5	Time 6	Time 7	Time 8
X4	Time 4	Time 5	Time 6	Time 7	Time 8	Time 9
X5	Time 5	Time 6	Time 7	Time 8	Time 9	Time 10
X6	Time 6	Time 7	Time 8	Time 9	Time 10	Time 11

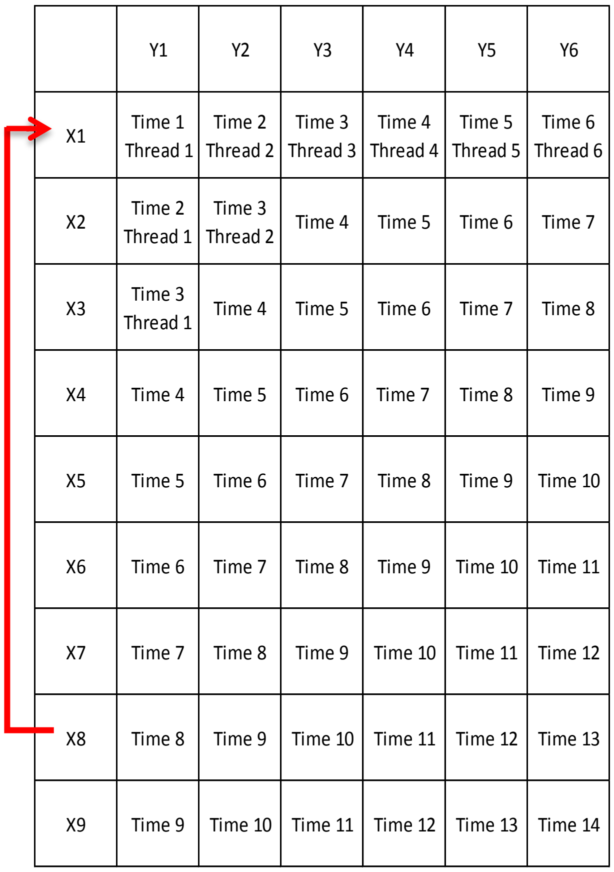
Figure 6 Parallel execution using threads

A simple solution was adopted by using dynamic allocation of a pool of memory, where just a few rows of the matrix are initially allocated, as seen in Figure 7, and the low index rows are reutilized as the wavefront advances through the array. In other words, when row X8 in Figure 7 needs to run, it reutilizes the memory initially allocated for row X1, since the results for X1 have already been propagated to row X2, and X1 is no longer needed. In the example mentioned earlier, involving 200,000 character documents, and assuming six threads running in parallel the initial memory allocation requirement becomes only 6.4 Mbytes. A larger number of threads will require more memory and synchronization while increasing the parallelism and reducing computational time.

5 EXPERIMENTS

To be able to properly test the proposed solution in this research, it was very important that the test data results could be verified, according to its known evolutionary sequence. Unfortunately, textual evolution has usually been recorded based on when the variant became public by being published or inscribed in some document, such as a letter to a friend or a dated draft kept in the author's files, and not work modification. Therefore, the most recently dated version must always be considered a version of one that was dated before it. Most of the dates used are those in which the literary documents were published or recorded in some publication or digital archive.

Considering that the main focus of this research is in the field of textual criticism, the sources chosen were of a literary nature and a realistic representation of works that would be studied by a literary scholar. These sources can be categorized in three different groups. The first group is comprised of documents, specifically poems, that were written and published in books and later digitized for Internet accessibility, obtained through the Elizabeth Barrett Browning (EBB) project at the University of North Dakota (UND), accessible online through UND's implementation of the Versioning Machine [10, 12]. These test cases range between having three to four versions beyond the original. The sizes of these poems are in a range of 508 to 2,699 characters each.



	Y1	Y2	Y3	Y4	Y5	Y6
X1	Time 1 Thread 1	Time 2 Thread 2	Time 3 Thread 3	Time 4 Thread 4	Time 5 Thread 5	Time 6 Thread 6
X2	Time 2 Thread 1	Time 3 Thread 2	Time 4	Time 5	Time 6	Time 7
X3	Time 3 Thread 1	Time 4	Time 5	Time 6	Time 7	Time 8
X4	Time 4	Time 5	Time 6	Time 7	Time 8	Time 9
X5	Time 5	Time 6	Time 7	Time 8	Time 9	Time 10
X6	Time 6	Time 7	Time 8	Time 9	Time 10	Time 11
X7	Time 7	Time 8	Time 9	Time 10	Time 11	Time 12
X8	Time 8	Time 9	Time 10	Time 11	Time 12	Time 13
X9	Time 9	Time 10	Time 11	Time 12	Time 13	Time 14

Figure 7 Memory allocation, row X8 reutilizes row X1

The second data set, represented as "WS" for William Shakespeare in Table 1, was obtained from the Internet Shakespeare Editions (ISE) website that is supported by the University of Victoria, Friends of ISE, and the Social Sciences and Humanities Research Council of Canada [2]. This dataset is made up of the quartos, folios and modern published versions of writings by William Shakespeare. This dataset consists of documents that range in size from 2,812 to 144,878 characters each. In particular, the data set labeled *Hamlet* was redefined in four different subsets to allow for an in-depth examination of the obstacles to establishing the evolutionary sequence of the text. The original *Hamlet* set contains 7 texts, while the remaining subsets contain only 3 to 5 files.

Lastly, a third test set is the poem "*Song of Myself*" found in the collection of poetry, *Leaves of Grass*, written by Walt Whitman and found in the Walt Whitman Archive housed by the University of Nebraska, Lincoln [11]. The reason of using this third case was to have a second set of data with more than five files. This dataset consists of documents that range in size from 70,239 to 73,473 characters each.
Below is a discussion of the computational time results associated with running the sequential and the multithreaded algorithms with the use of four threads in the parallel implementation. Table 1 shows the significant performance improvement obtained by the multithreaded configuration. The most significant gain, due to the size of the test case, was registered when running the William Shakespeare *Henry V* test case, which showed a speed up factor of 3.95 on the total execution time of the algorithm when compared to the sequential execution [1]. The smaller test cases showed minor improvements. In such cases, the overhead of activating the threads added to the synchronization delays between threads resulted in the smaller reduction of the execution time when compared to the sequential cases.

One of the main goals of this experiment was to verify that this technique can be used as a guide to determine the evolutionary sequence of versions of documents. In order to be able to determine the accuracy of the proposed method, a numerical scoring system had to be applied. Woon and Wong proposed the use of a new scoring system, restricted to graphs consisting of a single linear path and establishing windows of comparison along such path, which allowed one correct result to be counted multiple times according to the size of the window when a node preceded any of its actual successors in the path [16].

In this study, a window of size one was assumed, where an original is checked only against its immediate succeeding version, so a one point score is awarded to every edge that correctly matches the order of the actual text dates and zero points to those edges that fail the match. Table 2 shows the accuracy of the GRALT algorithm working with the MST and the SPE path options in the benchmark tests.

Table 1 Computational time for each test case.

Test Case	Sequential Processing Time (in seconds)	Multi-thread Processing Time	Multi-thread/Sequential Processing Speedup
EBB Child	0.898	0.515	1.74
EBB Bettine	0.566	0.343	1.65
EBB Sea	0.766	0.421	1.82
EBB Loved	1.092	0.390	2.80
EBB Clouds	1.282	0.609	2.11
EBB Dog	1.045	0.390	2.68
EBB Mitford	0.237	0.109	2.17
WW Leaves	1599.000	432.734	3.70
WS Hamlet	7224.080	2247.050	3.21
WS Henry IV Part I	3.354	0.905	3.71
WS Henry V	13.354	3.385	3.95
WS Richard II	4.321	1.201	3.60
WS Richard III	2.371	0.687	3.45
WS Romeo & Juliet	1.950	0.686	2.84
WS Troilus and Cressida	4.462	1.233	3.62

Table 2 Accuracy of GRALT with MST and SPE implementations

Test Case	Minimum Spanning Tree	SPE
EBB Child	100%	100%
EBB Bettine	66%	66%
EBB Sea	33%	33%
EBB Loved	100%	100%
EBB Clouds	100%	100%
EBB Dog	50%	50%
EBB Mitford	100%	100%
WW Leaves	100%	100%
WS Hamlet	66%	66%
WS Hamlet (exclude Quarto 1 files & Modern versions)	100%	100%
WS Hamlet (exclude Modern Versions)	66%	66%
WS Hamlet (exclude Quarto 1 & Modern Quarto 1)	75%	50%
WS Henry IV Part I	100%	66%
WS Henry V	66%	66%
WS Richard II	100%	100%
WS Richard III	100%	100%
WS Romeo & Juliet	100%	100%
WS Troilus and Cressida	100%	100%

6 CONCLUSION

In the analysis of literary works from the same author, it is important to know the evolutionary sequence, or versioning sequence, of the texts. In many situations, the actual sequence is unknown. The result of this study was the design and implementation of a system that can automatically make an estimation of the evolutionary sequence of digitized data that has not had its evolutionary derivation information recorded. The solution was based on the use of graph theory, in particular by embedding modified implementations of the Kruskal's Minimum Spanning Tree (MST) and Hamiltonian path (SPE for Single Path Evolution) algorithms to make the final determinations, which resulted in the design of the GRALT algorithm. The graph techniques were applied, resulting in a complete system design, which utilized a parallel wavefront implementation of a modified Levenshtein's algorithm to calculate text editing distances. The experiments showed a significant improvement on execution time when running a multithreaded implementation. There are still other approaches to be further investigated, including the improvement of the algorithm implementation computational time, which may be accomplished by increasing the number of threads and the number of processing elements or running on a distributed computer system.

7 REFERENCES

1. Hennesy, J.L., and Patterson, D.A., Computer Organization and Design The Hardware/Software Interface, Morgan Kaufmann Publishers Inc., San Francisco, CA, 1994.

2. (2013, March). Internet Shakespeare Editions [Online]. Available: internetshakespeare.uvic.ca.

3. (2013, June). Juxta[Online]. Available:http:// www.juxtasoftware.org.

4. Lamport, L. "The Parallel Execution of DO Loops," in the *Communications of the ACM Special Interest Group on Programming Languages*, February 1974, pp. 82-93

5. Levenshtein, V. I. (1966). Binary codes capable of correcting deletions, insertions, and reversals. *Soviet Physics Doklady*, 10(8):707–710.

6. Mayfield, K., Grant, E., Alberts, C. "Determining the Evolutionary Sequence of Versions of a Text with the Minimum spanning Tree Algorithm," Proceedings: 25th International Conference on Computer Applications in Industry and Engineering, 2012.

7. Mayfield, K. Alberts, C. "GRALT: Graphical Analyzer of Literary Text Versions," in the North Dakota Quarterly, Vol. 79, No 1, 2014.

8. Neapolitan, R., Naimipour, K. "Foundations of Algorithms Using C++ Pseudocode," Jones and Bartlett, Inc., New York, 2004.

9. Shillingsburg, Peter L. *From Gutenberg to Google: Electronic Representations of Literary Texts*. Cambridge, UK: Cambridge University Press, 2006.

10. (2011, November 15). The University of North Dakota: Elizabeth Barrett Browning Project [Online]. Available: http://und.edu/instruct/ sdonaldson/index.html.

11. (December 2013). The Walt Whitman Archive [Online]. Available: http://www.whitman archive.org/.

12. (2013, January 15). Versioning Machine v.4.0 (2011)[Online]. Available: http://v-machine.org/index.php.

13. Vetter, L., and McDonald, J. "Witnessing Dickinson's Witnesses," in *Literary and Linguistic Computing*, Vol. 18, No 2, 2003, pp. 151-165.

14. Williams, W.P., Abbott, C.S., "An Introduction to Bibliographical and Textual Studies." The Modern Language Association of America, New York, 1985.

15. Wise, M. "YAP3: Improved Detection of Similarities in Computer Program and Other Texts," in the *Proceedings of the ACM Special Interest Group on Computer Science Education*, 1996, pp. 130-134.

16. Woon, W.L., Wong, K.D. "String Alignment for Automated Document Versioning," Knowledge Information Systems, 2009, pp. 293-309.