Defragmenting Textual Data by Leveraging the Syntactic Structure of the English Language

Nathaniel Hayes
Department of Computer Science
Simpson College
701 N. C. St. Indianola, IA, 50125
nate.hayes@my.simpson.edu

Abstract

This paper examines the applicability of leveraging contextual heuristics embedded in the English language to properly reorder English plaintext files whose content has been fragmented into smaller segments, whose order have been shuffled.

Using only the textual content of each fragment, the presented algorithms determine how to reorder the segments, reconstituting the original file. This is achieved by analysing large quantities of published data to recognize patterns and heuristics that exist in English. These heuristics are then applied to each fragment to infer their order.

This approach to data reassembly shows considerable promise. Experiments show that plaintext split into five or fewer segments can likely be reconstituted perfectly. Even with more than forty distinct fragments, roughly 40% of the segments can be reordered correctly.

Introduction

The keynote speech at the Midwest Computing Symposium in 2016, given by Dr. Sarah Diesburg, addressed various security implications that accompany data storage. In this presentation, the speaker gave a general overview of what happens to data on a hard disk or NAND flash drive once data is deleted. When a file is “deleted” the operating system typically would mark the space occupied by the file as “free space” without actually zeroing out the data comprising the file.

This process is akin to a lazy landlord trying to evict a tenant. The landlord would send the eviction notice, and mark in his ledger that the space is free, but without actually enforcing the vacation of the property. He would operate as if the property were available, until he needs to move another tenant in. Although the landlord says the place is free, a simple knock on the door at that address will reveal the original tenant. The same applies to data on a disk, once a file is deleted the bits comprising the file content are left intact, only being changed once another file overwrites that section of data.

It is important to point out that files are not required to be stored sequentially on a disk. The files may be broken up into smaller segments and each segment can be stored anywhere on the disk. This is known as file fragmentation. A file information table on the disk specifies which pieces of data on the drive belong to the requested file *and in which order those segments need to be arranged* to reconstitute the file.

I walked away from the presentation wondering how plausible it is to recover a deleted file using only the raw, un-zeroed data. Obviously data recovery software exists to recover deleted files, but these pieces of software would typically employ methods attempting to reconstitute the original file metadata. My research, however, attempts to recover data in a theoretical worst case scenario, having no file metadata, working with arbitrary file types, and with the fragments of the file(s) being shuffled and disordered.

The problem my research attempts to solve is not dissimilar from a "word scramble," except using fragments of a file rather than letters. The fragments need to be reordered so that the file can be read correctly.

Theory

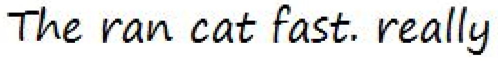

The ran cat fast. really

Figure 1: A scrambled sentence.

Take a moment to study the words in Figure 1. Generally, an English speaking person could fairly easily deduce that the original, sentence was "The cat ran really fast." We, as humans, are able to determine the original ordering of these words because we understand what a proper sentence looks like. Our knowledge of English grammar (syntax) provides us with heuristics like:

1. Periods should follow the last word of a sentence.
2. Words with capital letters are typically used at the beginning of sentences.
3. Verbs typically follow nouns.
4. Nouns typically come after the word 'the'
   ...

Suppose that the text in Figure 1 were in a file, and each word is a fragment of the file. Using just the text provided, and no additional information regarding the unshuffled state, a human using these heuristics can defragment this file with a high degree of accuracy.

Thus, I theorize that:

> If a file whose content has a definable syntax is fragmented into smaller segments, and these segments are not in their proper order, then with a suitable understanding of the aforementioned syntax the segments may be reordered.

While the example given in Figure 1 is English plaintext, the idea is not limited to English, or even human language at all. For example, a source file in Python has a very well defined syntax, with its own heuristics that can be used to deduce a sequential order:

1. Variables must be declared before they are accessed.
2. Line indentations occur after control structures (‘if’, ‘for’, etc).
   ...

While combinations of several heuristics can produce complex behavior, the intuitive simplicity of individual heuristic rules is notable. These heuristic rules are not explicitly taught, or hand-coded by a developer. The heuristics are simple relational patterns, learned automatically during training.

> If, during training, A is often found directly after B, then, during operation, if A and B are found, A will be placed directly after B.

This is a simplified explanation for how heuristic rules are used when determining the relative order of the data fragments. In practice, using heuristic rules is more complicated than this, since several rules may conflict. This process of heuristic application is fleshed out more fully in the Methodology section.

Methodology

The software developed during this research was designed to unshuffle fragments of an English text file. This is easier to debug and demonstrate than using multiple files or binary data. For the remainder of the paper, only English text is examined, however, software can be developed for other data types using the same methodologies.

Learning syntax

The first step in correctly unshuffling a fragmented file is understanding its syntax; learning what rules are present for that type of data. The algorithm used to learn a file’s syntax is derived from observing how humans learn syntax.

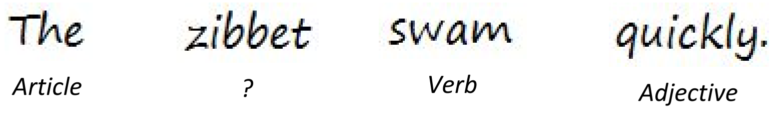


Figure 3: Demonstrates that people can identify what part of speech ‘zibbet’ is; despite it being a made-up word

After a quick glance at the text in Figure 3, most people would be able to readily identify the word ‘zibbet’ as a noun. This is interesting because zibbet is not a real word. Some process in the brain was able to determine ‘zibbet’ is a noun without ever having been exposed to the word before. This is quite impressive.

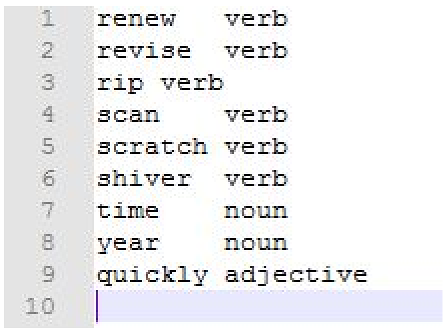
I believe this is because of what are known as “context clues.” The process of “looking at context” involves finding terms that are known, then making assumptions about unknown terms based on how the known terms have been used before. This is how people are able to understand the meaning of an obscure word in a book without having ever been given a formal definition. As Figure 3 shows, it also allows people to tag what parts of speech a word exhibits without having ever seen the word before.

> The emulation of the brain’s ability to learn from context clues is the backbone of the training process. The ability to use pieces of existing knowledge together to generate new information greatly reduces the need for human oversight in training.

Contextual Learning Theory

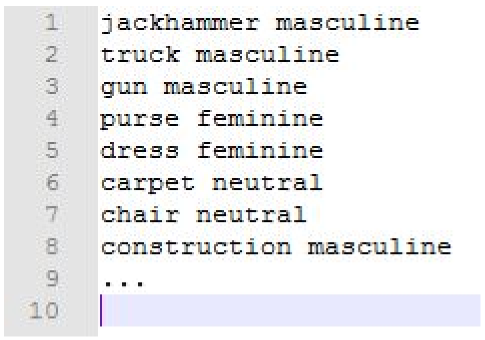
The algorithm developed to emulate this learning style requires entering a relatively small amount of data by hand. This is because the algorithm iteratively builds upon existing information, so it needs a starting point.

To emulate Figure 3, the initial information fed into the learning algorithm is a long list of words with their parts of speech, an example of which can seen in Figure 4.1. This initial information is the training data from which the algorithm will learn what characteristics distinguish these parts of speech. In my research the training data consisted of 15 books including all three Hunger Games books, Charles Darwin’s *The Origin of Species*, The King James Version Bible, and even some children's stories.



```
renew   verb
revise  verb
rip verb
scan    verb
scratch verb
shiver  verb
time    noun
year    noun
quickly adjective
```

Figure 4.1: Initial data; algorithm would learn to classify a word's part of speech.



```
jackhammer masculine
truck masculine
gun masculine
purse feminine
dress feminine
carpet neutral
chair neutral
construction masculine
...
```

Figure 4.2: Initial data; algorithm would learn to classify a word's. grammatical gender.

There are two primary genres of machine learning algorithms: regression and classification. A regression algorithm is applied in situations requiring an output of numeric continuous values, such as predicting tomorrow's temperature or evaluating how well a particular stock will do. These algorithms output numbers. Classification learning algorithms on the other hand are used to determine what classification an input would likely fall into. Some examples could be determining what tomorrow's forecast will be (rainy, sunny, cloudy, etc,), or whether a stock should be bought or not (buy it, don't buy it).

The learning algorithm applied to learn language syntax is a classification algorithm, it categorizes terms so that patterns can be found. It is given a set of categories and terms that belong to those categories. During training it sees how these terms are used in context with their neighbors; then it applies those findings to other terms that are used in similar ways.

Figure 4.2 is used to illustrate this. Given the categories "masculine," "feminine," and "neutral," along with words which belong to these categories, the algorithm would see how these words are used and infer that other words used in similar ways likely fall into the same categories.

> The algorithm using the data from Figure 4.2 knows that "dress" is feminine. If the training input contained the sentence "*That dress is pretty*, yet *that belt is pretty* also" the algorithm would infer that "belt" is also feminine because it is used in a similar way to "dress" in this context.

Once information is able to be inferred about a previously unknown term the term is added to the algorithm's internal dictionary for future inferences to be gleamed.

> This has a compounding effect. Continuing the previous example, if later, the sentence "*That belt* does not go with *that hat*!" were found then "hat" could be classified as "feminine" because its usage is similar to that of "belt."

The data initially fed into the learning algorithm includes a list of the 500 most common verbs in the English language, the 500 most common nouns, the 100 most common adjectives, etc. The file has at least ten examples for each part of speech, however the file altogether is comprised of less than 2000 words. This information needs to be provided earlier in the paper. This might seem like a lot, but after the learning algorithm is done processing its training data of 15 books it has encountered over 200,000 words. This disparity between the number of known words and of unknown words causes the algorithm to infer large portions of its dictionary, which has negative consequences.

As versatile as this process can be, it has drawbacks. With each iterative inference, accuracy is reduced. The further down the chain the algorithm needs to look to make an inference the less likely that inference will be true. This is why it is important to have a large set of initial data, having more terms that can be learned directly from the input data will result in more accurate inferences.

Contextual Learning Implementation

The algorithm must remember how each term in the training data is used, and then apply this to future terms that are used in a similar way. During the training process a very large amount of data is computed detailing how often each part of speech occurs in the three words before and after every word. This is essentially a frequency tally for each word in the dictionary with respect to how often each part of speech is found nearby the word. Initially it can only recognize the parts of speech of words given in the initial data, but as the algorithm progresses through the training data it is able to use this inference in tallying frequencies.

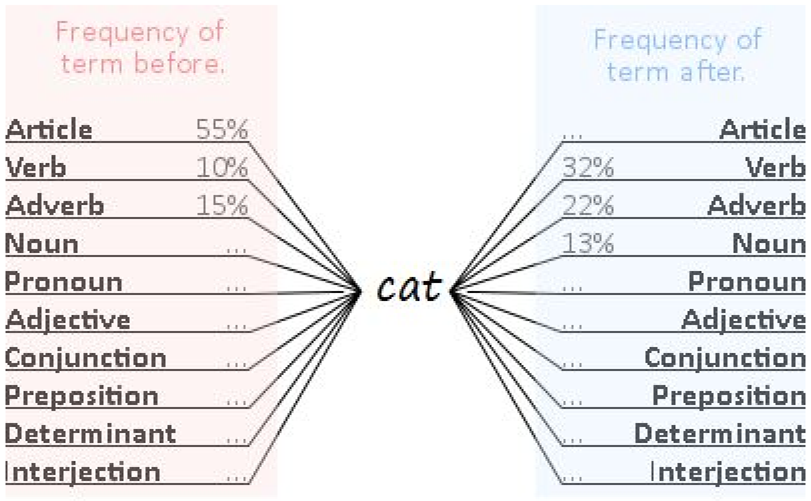


Figure 5: Represents the kind of data collected for each word encountered during training. For simplicity, statistics are shown for the words immediately before and after. The algorithm actually computes statistics for three words on either side.

This provides the necessary information required to build and recognize patterns. For example, using Figure 5, this information could allow the algorithm to take the word "cat" and see that it is typically preceded by an article and followed by a verb, such as in "The cat ran."

> Knowing that the term T typically comes after category C_x and before category C_y allows the algorithm to tentatively infer the category, C_x or C_y, of any term adjacent to T.

For example, if we know that nouns typically come directly after the word "the" then if we find a word that we do not recognize, but comes directly after "the," it can be inferred that the word is a noun as in Figure 3.

The heuristics for English are created using this information. Every word has its own heuristic, or pattern, for which kinds of words should surround it.

Part of Speech Tagging Theory

At this point the algorithm would have a very substantive set of heuristics, but these cannot yet be applied to a fragmented file. This is because the words in the fragments have not been tagged with their parts of speech yet. The algorithm needs to know the part of speech of all the words in the fragments so that the patterns can be applied.

Note that the part of speech of a word is determined using the frequency statistics of surrounding words obtained from the training data. Counterintuitively, however, the actual word that is being tagged is not usually not considered in determining that word's part of speech.

This is because a word can have multiple parts of speech depending on how it is used, as shown in Figure 5.

Give me a hug!
Noun
Hug me!
Verb

Figure 6: Example how the same word can
have different parts of speech, depending on context.

As in Figure 6, knowing that the word to be tagged is "hug" is not sufficient to determine its part of speech. It is necessary to know the context of the word, which comes from the training data. The training data could easily show that nouns come after the word 'a, ' and that verbs frequently come before the exclamation "me!"

> The same heuristics that are used to unshuffle a file's fragments are also used to classify the terms contained in those fragments.

This might sound strange, applying heuristics to the fragments so that heuristics can be applied to the fragments, but this process is similar to what was discussed before. When the algorithm is making inferences on how fragments should be ordered, it is basing those decisions on prior inferences made in tagging the text. This is exactly how humans do it. Looking back to Figure 1, we would first infer that "cat" is a noun, and from that we would infer that "cat" should be placed after "The".

Part of Speech Tagging Implementation

For simplicity, I have been giving examples of heuristics that apply to terms that are directly adjacent to each other, such as "nouns typically come directly after 'the'." The algorithm actually records the frequencies of words up to three terms away. This allows for heuristics that span a larger distance. For example, and this is a real heuristic found during training, the third word prior to "aberrant" is typically a verb.

Usually, the further apart two words are from each other the less impact they will have on each other, and thus the less often any heuristic spanning that distance will hold true. However, using a scope slightly larger than direct adjacency increases the number of data points being used for inference, from two to six; allowing the algorithm to make inferences over a longer distance.

Referring back to Figure 5, note how the information recorded during training is given in the form of twenty percentages. A heuristic that would be derived from this, as stated earlier, is "Articles typically come before the word 'cat'." This is derived because, during training, articles have been found before the word 'cat' 55% of the time, more frequently than any other part of speech. Therefore the heuristic would be accurate, but using only that heuristic would result in false-positives 45% of the time. This is why multiple heuristics need to be used in tandem.

> When tagging text there will always be more than one heuristic considered for every term that needs to be tagged. There will be the heuristic from the previous term which applies to the word following it, and there will be the heuristic that the next term applies to the word before it

Words on both sides of the current term will influence how it is tagged. The task is to take the available heuristics, aggregate them, then use this aggregated data to infer what part of speech the current word should be. This aggregation is implemented by averaging the percentages given by the adjacent terms, weighed in conjunction with sample sizes, with respect to the ten parts of speech. The term would be tagged with whichever part of speech has the highest value after this.

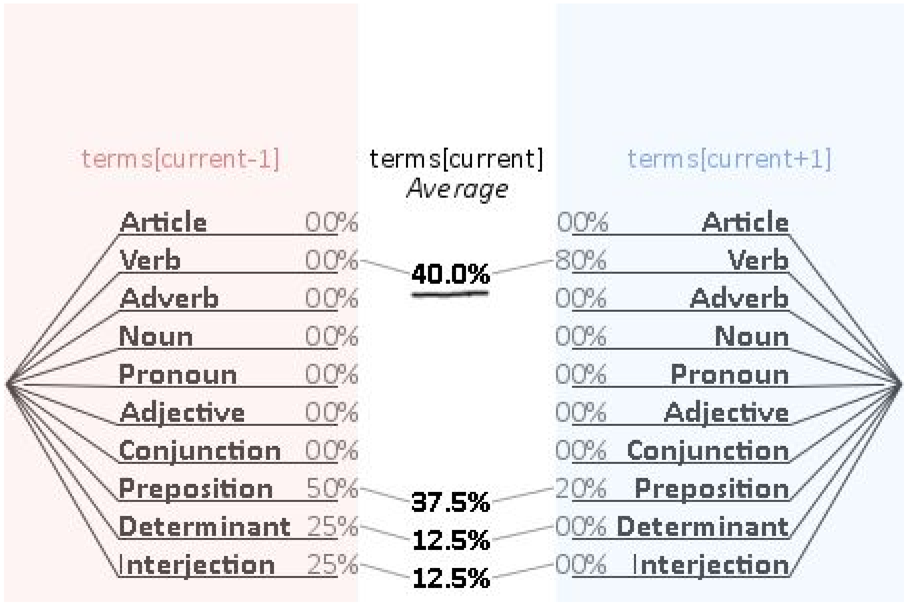


Figure 7: Illustrates the process of aggregating frequency data to tag parts of speech.

In Figure 7, even though both the red and blue words suggest that the current term could be a preposition, the blue term's "the previous word is probably a verb" heuristic is accurate enough to supercede the red term's "the next word is probably a preposition" heuristic.

The same process is used when leveraging additional terms beyond those directly adjacent to the current word, but with a few more steps. The more distant terms are used to help make a decision when closer pairs of words produce conflicting result; they act as tiebreakers. Their data is aggregated in in the same way the adjacent pairs are, but with increasingly smaller weights relative to their distance.

Unshuffling fragmented text

The only thing left to do is take all of the learned heuristics and apply them to the file fragments' tagged text to produce a sequential order for the fragments. This is fairly straightforward.

Think about a blank white puzzle, with all the pieces mixed up on the floor. The puzzle can still be solved, even with no picture for guidance. Any piece will give some idea of what the surrounding pieces should look like because of the indentations those adjacent pieces have made in the current piece. Slowly, as pieces are successfully placed together fewer pieces need to be tested, quickening the pace at which the puzzle is solved. The same thing is done to reassemble a fragmented file, except instead of physical indentations the algorithm uses the learned heuristics to find matches.

The fragments can be ordered by applying heuristics to the edges of each fragment and testing which other fragments fit best. For example,, suppose one fragment ends with the word “The” and another fragment starts with the word “cat.” The fragment starting with “cat” would be placed after the fragment starting with “The” because nouns typically follow the word “the.”

As mentioned above, the algorithm can learn and use heuristics which look further than just adjacent terms, up to three words on either side. This increases the perceivable indentations at the edges of each fragment, improving chances of accurately finding matches. Figure 8 illustrates the indentation left by predicting up to three terms ahead.

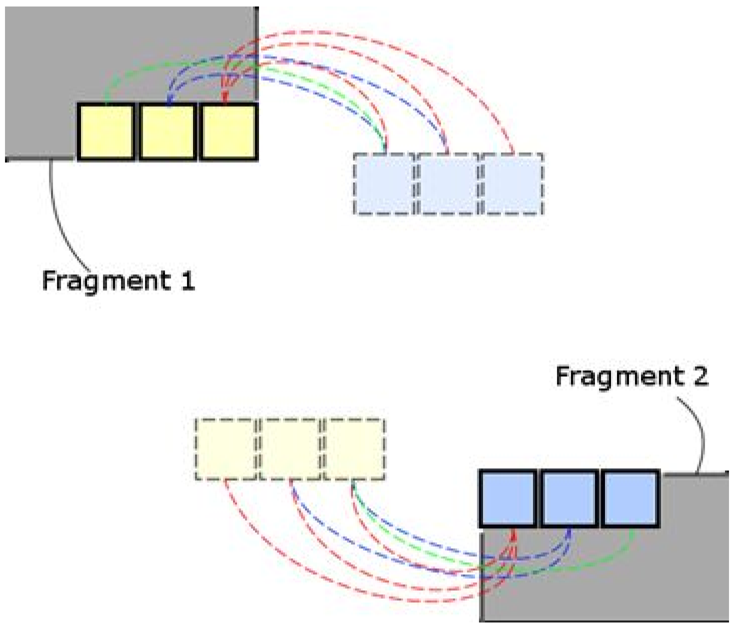


Figure 8: Illustrates how applying heuristics to fragments can act like the indentations in puzzle pieces.

Results

A demonstration of the part of speech tagging algorithm can be seen in Table 1.

Input	Output	Accuracy
Here are a few examples of what this baby can do.	here - Adverb are - Verb a - Article few - Determinant examples - Noun of - Preposition	92%

	what - Determinant this - Pronoun baby - Noun can - Pronoun do - Verb . - Punctuation	

Table 1: Sample output displaying high accuracy.

Noting the example in Table 1, of the nearly two-dozen hand checked tests, the tagging algorithm has never had an accuracy below 80%, and is frequently in the mid-to-low nineties. While these results are impressive, these numbers should not be misunderstood as ground breaking. At the time of writing, a good part of speech tagger would obtain accuracies upwards of 97%[1].

The goal of this process is to reconstitute a fragmented file. If this method suggested a new arrangement of fragments are no closer to being properly ordered than the original input then the method would have failed. However, if the suggested arrangement of the fragments exactly reconstitutes the original file then the method would have succeeded perfectly. Thus, this method is evaluated by what proportion of the file's fragments are correctly ordered after processing.

This is measured not by how many fragments are in their correct absolute position, but instead by how many fragments are in the correct relative position to their neighbors. Of course, getting each fragment into the correct absolute position is necessary to reconstitute the file, but if each fragment is in the correct relative position then each fragment will also be in the correct absolute position.

For example, suppose the correct order for the fragments should be {5, 2, 3, 4, 1} yet the defragmentation algorithm produced a recommended sequence of {2, 3, 4, 1, 5}. None of these items are in the correct absolute location, but the first 4/5 of the sequence is still unshuffled. This is taken into account when a sequence is being evaluated.

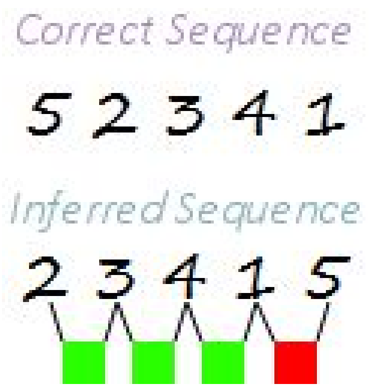


Figure 9: Demonstrates how a reassembling sequence is scored.
The red squares indicates the fragment pair which is not in the correct order,
the green squares indicate fragment pairs which are in the correct order.

This sequence would get a score of 75% correct.

Figure 10 shows the results of applying these algorithms to *The Communist Manifesto*, split into 41 evenly sized, shuffled, fragments.

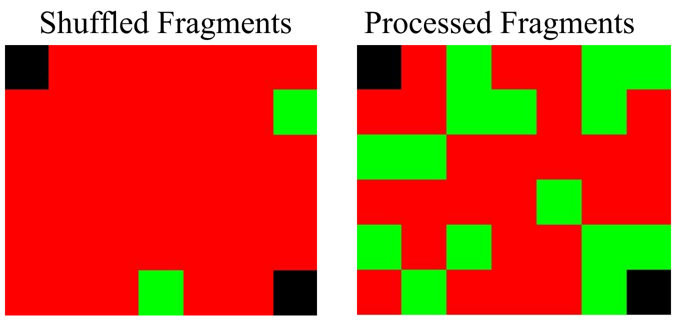


Figure 10: The results of attempting to defragment *The Communist Manifesto*. Red squares indicate fragment pairs which are not in relative order, green squares indicate fragment pairs which are in relative order. The black squares represent the beginning and ends of the files, where no link between fragments exists.

When running the experiment depicted in Figure 10, after the fragments were shuffled amongst themselves two pairs of fragments were already in the correct order, as can be seen from the two green squares in the “Shuffled Fragments” image. However, after applying the method to this data, the resulting file has fifteen pairs of fragments ordered correctly, as can be seen in the second image. Fifteen pairs out of forty yields a 37.5% correct reconstruction.

This process works better with fewer fragments, as can be seen from the graph in Figure 11.

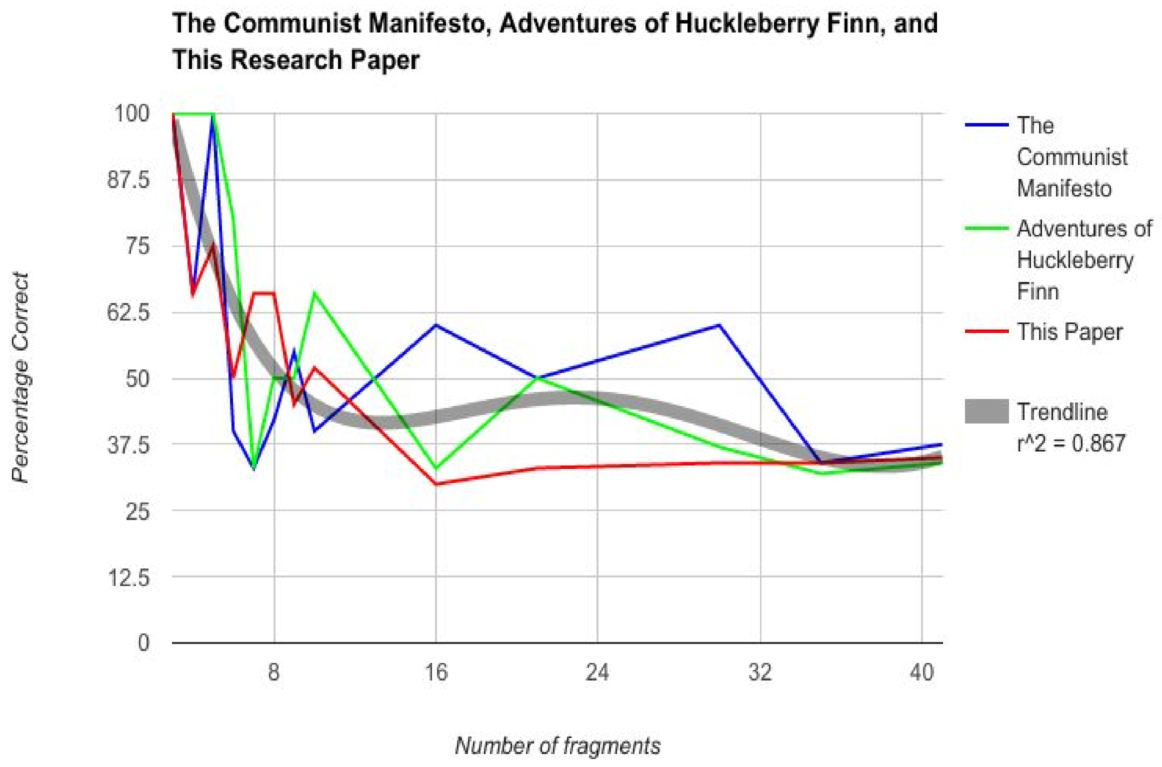
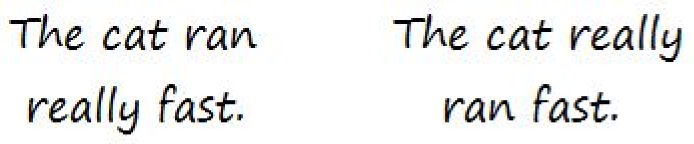


Figure 11: Depicts the performance of the algorithm as the number of fragments increases. Notice how the performance of this algorithm decreases as the number of fragments a file is split into increases.

These results are very promising, and they provide strong evidence to support the theory that contextual heuristic analysis can be used to reconstitute fragmented files.

Limitations

The most glaring limitation posed by this theory is that perfect reconstruction *cannot* be guaranteed, regardless of how well the underlying syntax is understood. The reason for this can be seen in Figure 11.


The cat ran really fast.

The cat really ran fast.

Figure 12: Two possible, and equally legal reconstructions of the shuffled words in Figure 1.

Since this theory is attempting to emulate human cognition it has the same limitations. Both of the sentences in Figure 12 are perfectly valid sentences according to English syntax. Even if it could be deduced that the first sentence is far more likely, there is *always* a non-zero chance that the second sentence is correct instead. Human defragmentation is limited in the same way.

As was discussed earlier, another pressing limitation comes from the initial training data needed to properly train the algorithm. The more information is given to the training algorithm initially the the more effective training will be, and insufficient data will produce inaccurate results.

Acknowledgments

I am deeply thankful to Professor Mark Brodie and Professor Lydia Sinapova for their comments and editing assistance that greatly improved this manuscript.

References

1. Manning, Christopher D. "Part-of-Speech Tagging from 97% to 100%: Is It Time for Some Linguistics?" Computational Linguistics and Intelligent Text Processing Lecture Notes in Computer Science (2011): 171-89. Stanford. Web.