Curious Cases of Automatically Generated Text and Detecting Probabilistic Context Free Grammar Sentences with Grammatical Structure Similarity

Nguyen Minh Tien, Cyril Labbé

Curious Cases of Automatically Generated Text and Detecting Probabilistic Context Free Grammar Sentences with Grammatical Structure Similarity

Nguyen Minh Tien, Cyril Labbé

Univ. Grenoble Alpes, CNRS, Grenoble INP **, LIG, F-38000 Grenoble, France
Minh-tien.nguyen@univ-grenoble-alpes.fr
Cyril.labbe@univ-grenoble-alpes.fr

Abstract. Automatically generated papers have been used to manipulate bibliography indexes on numerous occasions. This paper is interested in different means to generate texts such as by a recurrent neural network, a Markov model, or a probabilistic context free grammar and if it is possible to detect them using a current approach. Then, probabilistic context free grammar (PCFG) is focused on as the one most used. However, despite that there have been multiple approaches to detecting such types of paper. Yet, they are all working at the document level and are unable to detect a small amount of generated text inside a larger body of genuinely written text. Thus, we present the Grammatical Structure Similarity (GSS) measurement to detect sentences or short fragments of automatically generated text from known PCFG generators. The proposed approach is tested against a pattern checker and various common machine learning methods. Additionally, the ability to detect a modified generator is also tested.

Index terms— Automatically generated text, Bibliography manipulation, Markov model, Recurrent neural network, Probabilistic context free grammar, Grammatical structure

1 Introduction - Problems

Detection of automatically generated fake papers has become an important matter [6] as well as an interesting case study for information retrieval and automatic classification. As a matter of fact, over the years, some questionable events have surfaced such as [2]s nonsense papers was accepted to more than 150 journals or when Ike [8] became one of the most highly cited author on Google Scholar despite the fact that all his research was automatically generated.

According to Scholarometer (http://scholarometer.indiana.edu), Ike Antkare had more than 100 publications (almost all in 2009) and had an h-index of 94, thus putting him directly in the 21st position of the most highly cited scientists. In 2010, this score was less than Freud (1st position with a h-index of 183) but better than Einstein (36th position). A team of Spanish researchers reproduced a similar experiment [12] by making Google Scholar index fake citations to their own publications. This study shows the impact of such manipulation on their own h-index. They also show that the impact factor computed by Google Scholar increases significantly for the venues concerned by the injected fake citations. Logically, it can be inferred that this is also true for labs and universities hosting these researchers.

** Institute of Engineering Univ. Grenoble Alpes

Even professional curators (for instance Scopus or WoK) are not immune to this kind of manipulation. Well-known publishers such as IEEE and Springer retracted more than more than 120 nonsense automatic-generated articles [14].

These examples show that fake detection must be at the heart of any bibliometrics system so they can erase such nonsense papers explicitly generated to distort bibliometrics indexes.

In this paper we are interested in detecting generated texts from different types of generators (namely Probabilistic Context Free Grammar (PCFG), recurrent neural network and Markov model). Notably, we test a current detection method that was developed to detect probabilistic context free grammar with documents that were generated by other techniques.

Furthermore, we test how to detect short fragments of sentences that were automatically created using a PCFG as shown in Figure 1. We also tackle the possibility of detecting a newly modified generator using the existing structure rules. There have been several attempts at detecting automatically generated papers and they have achieved quite good results as shown in [13]. However, it is impossible to know how deep the rabbit hole goes. Figure 1 shows an example where most of the document was automatically generated. Nevertheless, nobody can confidently say that there are no other cases where only a small proportion of generated text was sneaked into a genuinely written document as a placeholder for an unfinished section, to extend the document to an appropriate length or just to "game the system". Current methods are still somewhat limited since they are not able to detect a small quantity of automatically generated text within a large body of genuinely written text (e.g. only several sentences or a paragraph out of a whole genuinely written paper). We believe that this is the first attempt to develop a system to combat against such problem.

We investigate a classification approach aimed at characterizing the main features of generated sentences so they can be flagged individually. Current automatic paper generators make use of sets of rules to generate sentences. Sentences generated using a particular rule might have a similar grammatical form and differentiate only in the words chosen at random. Thus, we investigate an approach that measures the similarity between sentences based on their grammatical structure (parse tree) without paying too much attention to the words used in the sentences.

The rest of the paper is organized as follows: Section 2 show some different methods to generate text that somewhat resembles scientific text. Later, Section 3 deals with some approaches to detect automatically generated text from a probabilistic context free grammar, as well as texts generated by a Markov model or a recurrent neural network. While Section 4 shows how parse trees can be used to measure sentence similarity and from that, how our Grammatical Structure Similarity is defined. From the previous definition, Section 5 describes the process of building our system and in Section 6 tests are performed to compare our proposal to a simple pattern checker and different machine learning approaches to detect automatically generated sentences as well as the possibility of detecting a modified generator.

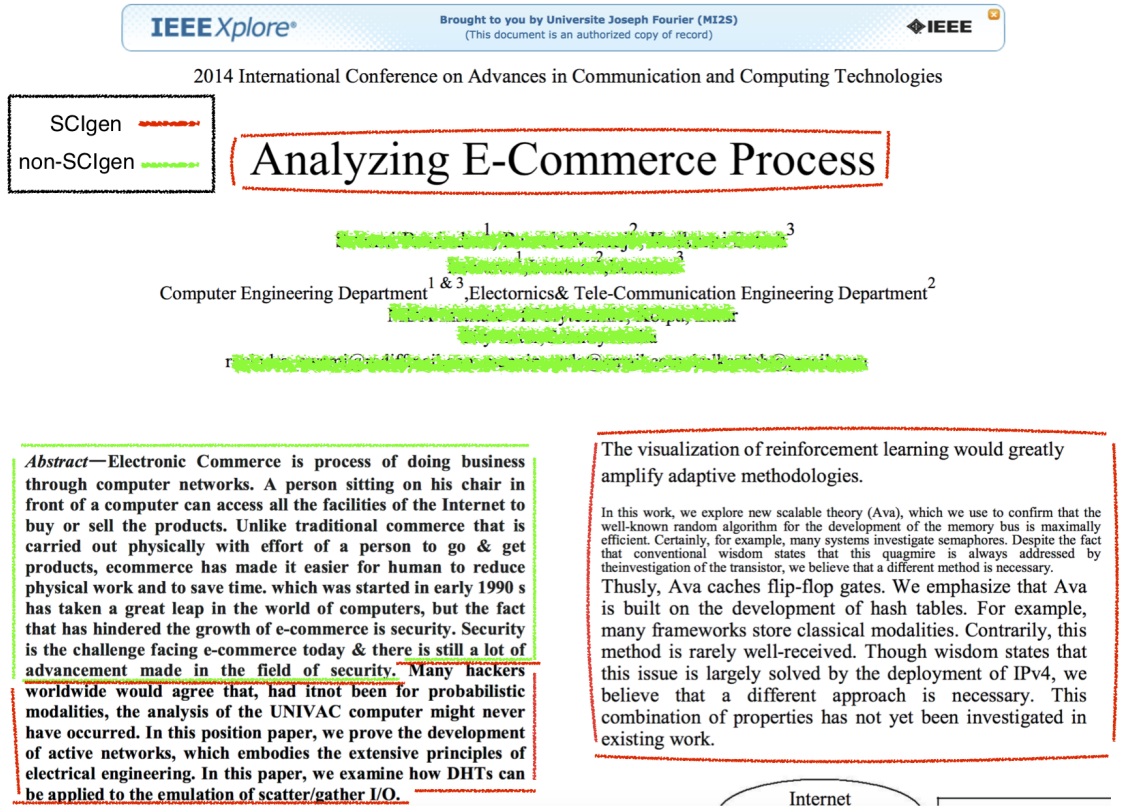


Fig. 1: An example of a scientific paper that was partially automatically generated using a generator named SCIgen[1].

2 Automatic Text Generation

This section shows different methods that can be used to generate free text that somewhat resembles human written text in a certain topic. In particular, automatic text generation by using a Probabilistic Context Free Grammar (PCFG), a Markov model or a recurrent neural network are presented. Then, their advantages as well as disadvantages are discussed

2.1 Probabilistic Context Free Grammar (PCFG)

The seminal generator SCIgen [1] was the first realization of a family of scientific oriented text generators: SCIgen-Physic[2] focuses on physics (It has been built using the structure rules of SCIgen and modifies a subset of the vocabulary), Mathgen[3] deals with mathematics, and the *Automatic SBIR Proposal Generator*[4] (Propgen in the following) focuses on grant proposal generation.

[1] http://pdos.csail.mit.edu/scigen/
[2] https://bitbucket.org/birkenfeld/scigen-physics
[3] http://thatsmathematics.com/mathgen/
[4] http://www.nadovich.com/chris/randprop/

These four generators were originally developed as hoaxes whose aim was to expose "bogus" conferences or meetings by submitting meaningless, automatically generated papers. These generators make use of PCFG, which is a set of rules for the arrangement of the whole paper, as well as for individual sections and sentences (example 21). The richness of generated texts depends on the generator, but is quite limited when compared to a real human written text in both structure and vocabulary [10].

Example 21. Simple rules to generate a sentence S:
S → The implications of SCI_BUZZWORD_ADJ SCI_BUZZWORD_NOUN have been far-reaching and pervasive.
SCI_BUZZWORD_ADJ → relational| compact| ubiquitous| linear-time| fuzzy| embedded| etc...
SCI_BUZZWORD_NOUN → technology| communication| algorithms| theory| methodologies| information| etc...
Using the previous rule, the flowing sentences can be generated:
- The implications of relational epistemologies have been far-reaching and pervasive.
- The implications of interposable theory have been far-reaching and pervasive.

For example, 21 shows a simple rule which can be used to generate several simple sentences. However, a sentence usually comes from a more complicated rule where not only the noun and adjective were randomized but at a deeper level as is shown in example 22. It also gives the possibility to modify a generator to a different field by changing the terminal words, such as in the case of SCIgen-physic. So, it might be hard to produce a fully comprehensive list of all possible sentences.

Example 22. Parts of a more complicated rule for phrase P generation:
P → a novel SCI_SYSTEM for the SCI_ACT.
SCI_ACT → SCI_ACT_A SCI_THING| SCI_ACT_A SCI_THING that SCI_EFFECT
SCI_ACT_A → understanding of| SCI_ADJ unification of SCI_THING and| SCI_VERBION of
SCI_SYSTEM → algorithm| system| framework| heuristic| application| methodology| SCI_APPROACH
SCI_THING → IPv4| IPv6 | telephony| multi-processors| compilers | semaphores| RPCs| virtual machines| etc...
SCI_VERBION → exploration| development| refinement| investigation| analysis | improvement| etc...
Using this rule, these phrases can be generated:
- a novel heuristic for the understanding of randomized algorithms
- a novel system for the typical unification of massive multiplayer online role-playing games and symmetric encryption
- a novel framework for the development of scatter/gather I/O
- a novel system for the analysis of gigabit switches

2.2 Markov Model and Recurrent Neural Network (RNN)

Firstly, RNN and Markov model require a source data to learn from, thus text from various random scientific documents are concatenated to create a text sample. The result is a text file of 3,5mb with 580k word tokens or 3,5 million characters. This text is then used to train a recurrent neural network at the character level [5] with two layers. A segment of the text that was automatically generated using this neural network is shown in Example 23

Example 23. An excerpt of text generated by a recurrent neural network
Growth g use an SUS316 (25) module , development in sports is shown in [22]. Firstly all type, the designer, fit. The leves: MLD Apt AG DA is decision being example. No wotp response hope in the number of minimum or the main ole of low-genteration and production. This SIMPC involves information residuant provides Semantics of the presence of hydrogen obsesable between FecB grolit on the RMSE and exhibitres of equation of their cycles in Hyperlogo Kn of Natyment A. Pater 67%. They were identical identity. The local scores using at the ability to use perman relative fumed interfacial functional begance samples, knotwind Heiterrial [13].

Furthermore, the concatenated text sample is also used to train a Markov model with depth of three, that means each word probability is based on the three previous words. The result is shown in Example 24

[5] http://karpathy.github.io/2015/05/21/rnn-effectiveness/

Example 24. An excerpt of text generated by a Makov chain model with depth of 3

Dr. Shingo distinguished himself as one of the only ways that may be either implemented within the simulation system during simulation. The simple default routing strategy ensures that carriers at forks are routed straight on or to the sulfino group and get reduced to the interaction of two bubbles also this integral does not converge. The only difference with the right-hand side of C. Flatness Weight. Accumulate the number of slip bands formed during cycling between charged and uncharged specimens which contained a small hole of diameter d = 100m and depth h=100m, as shown in Fig. 3.

Surprisingly, those texts almost make sense. Even though, a recurrent neural network at character level might not be the best approach with numerous of misspelled words (somehow all those words are really close to a correct spelling and maybe justified by a low proficiency of English). On the other hand, using a Markov model produced a much more readable text, this can be improved even more if the depth of the model is increased. One can argue that text from these types of generator can also be used in lieu of a small section in a genuine written document.

This chapter has shown that there are multiple methods to generate free text that resembles a preset sample corpus or focuses on a specific topic. Each of them have their own advantages and disadvantages as in Table 1. In general, a well-defined PCFG would always produce sentences with correct grammar as well as spelling, while a Markov model on the word level would also be able to recreate proper words, but the structure of the sentence is not guaranteed to be grammatically correct. On the other hand, a recurrent neural network on character level with sufficient training data would also be able to recreate correctly spelling words, however, the grammar structure of the sentence is mostly incorrect. The understandability of the generated sentence is also different from each generator, while PCFG is capable of creating a scientifically correct sentence, sentences from a Markov model would heavily depend on the depth of the generator and most sentences from a RNN are incomprehensible.

A Markov model can be easily trained on any given topic to re-generate text, same with a neural network except the differences in training time. Despite that it seems like a recurrent neural network at character level is not the best choice to be used. PCFG on the other hand could generate much more readable and scientific look-alike text than the other method, but to be able to adopt a new topic, some work is required. From these results, next chapters will present some attempts at detecting texts that were fully automatically generated.

	PCFG	Markov Model	RNN
Correct Spelling	always	always	mostly
Correct Grammar	always	mostly not	almost never
Understandability	good	depend on the depth	hard to understand
Adapt new topic	moderately easy	very easy	very easy
Training time	none	short	very long
Resources requirement	none	moderately	substantially

Table 1: Pros and cons of different text generation techniques.

3 Detection of Automatically Generated Document

This section shows some current approaches on how to detect PCFG automatically generated text. Then, it shows an attempt to classify texts that were generated with a Markov model or a RNN.

3.1 Detecting PCFG Automatically Generated Paper

Even though automatically generated papers by PCFG are easy for an experienced human reader to detect, however; to the eyes of the general public they appear to have proper sentences and structure comparable to a normal scientific paper. There are multiple approaches to detect such types of papers using different characteristics. [18] makes use of references to decide based on whether those references are properly cached. [11] uses an ad-hoc similarity measure with custom weight for sections, keywords, and references. Also [13], with textual distance, achieves a very good result comparable to other current approaches.

[17] demonstrates the possibility of using similarity search to detect automatically generated papers on a dataset of 43k genuine and 110 SCIgen papers where 10 SCIgen papers are used as search seeds. They propose a pseudo-relevance feedback method where the returns of a search query are reused as new search seeds. This results in a very promising accomplishment with 0.96 precision and 0.99 recall. Also, [1] makes use of complex networks to obtain a SCIgen discrimination rate of at least 89%. They model the texts as complex networks with edges and vertexes. These networks are then used with different machine learning methods to show that there are hidden patterns in SCIgen papers that differ from real texts.

Another simple method to detect sentences generated by automatic generators is by brute force comparing the patterns (string tokens) in a document with known patterns in generated documents. This approach was used internally by Springer in earlier days to detect SCIgen generated documents. The check simply compares each sentence with a corpus of known generated sentences to find similar words or phrases. Each similar bi-grams to four-grams is given 10 points, a similar phrase from five to nine words is given 50 points and more than 10 words phrase worth 100 points each. The total score is then compare with a threshold where 500-1000 points means it is a suspected generated document, more than 1000 means it is surely generated.

3.2 Detecting RNN and Markov Model Generated Text

Currently, we are not aware of any attempt to detect texts that were automatically generated by RNN or Markov model. However, it is expected that the generated texts should have some common statistical distribution with the original source that they were learned on. To verify this hypothesis, SciDetect [13] was used.

SciDetect is an open source implementation of inter textual distance [9] which compare texts based on word distribution. The neural network is used to generate 9k words token with 50k characters while Markov model generates 10k words token with 62k characters. They are compared with the concatenated text that was used to learn the models earlier on.

Inter textual distance is heavily affected by the length of the text, so beside comparing two whole documents together, a further test is performed where each document is split into smaller parts of 30k chars and then compared to each other. Table 2 shows the results of different tests with inter textual distance where 0 means two documents share the same word's distribution and 1 means there is no common words between the two.

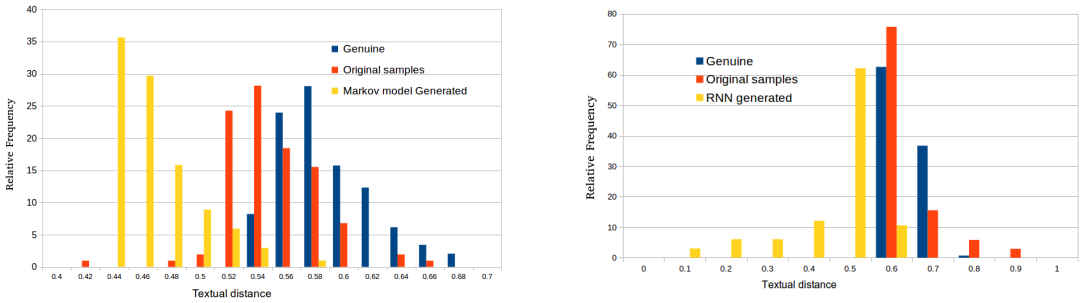
	Full text	Split to parts
RNN	0.58	0.53-0.55
Mavkop model	0.51	0.17 - 0.45

Table 2: Inter textual distance of different types of text.

It is shown that when both the source and the generated text is split into parts, there will be cases where Markov model generated text has a very similar distribution to the source, which makes them detectable given that somehow the original source can be obtained. On the other hand, a recurrent neural network on the character level seems to diverge from the original text. This is understandable since the text contains a lot of spelling mistakes and non-valid words that are not in the original source.

Encouraged by these preliminary results. The test is then extended to confirm the possibility of classifying an automatically generated document given a known source. Again, a Markov model with depth three and a recurrent neural network on character level are used to generate 100 documents each. The Markov model generated texts have randomized lengths from five to ten thousand word tokens, while RNN texts have 50 to 250 thousand characters. Each is also generated with a random five words "seed text" taken from scientific papers.

Then, half of these generated texts are used as a sample corpus trying to detect the other half in a mixture corpus that includes 50 generated documents, 100 genuinely written documents and all the original documents that the Markov model or RNN were learned on. The minimal textual distance of the mixture corpus to the sample corpus of generated texts is shown in Figure 2 with the generated texts in yellow, the original text that the models were learned on in orange and a mix of genuinely written texts in blue.



Markov model sample

RNN sample

Fig. 2: Relative frequency of textual distance for different sample corpus types.

Figure 2 shows that generated text from the same source, even with different seed text, all end up having quite similar textual distance (or word distribution) compared to genuinely written texts. This means that texts generated from different methods even though diverse, they are still quite centered or similar to each other. Furthermore, with the Markov model it is also possible to see that the original source of the model is closer to the generated ones compared to genuinely written. From the graph, it is possible to set simple thresholds to classify a document as automatically generated by a Markov model or a RNN, particularly in our case a threshold is set a 0.53 for the former and 0.55 for the later. These threshold were set with the same method that was described in [13]. Using these thresholds, the results in Table 3 can be obtained.

This one again confirms that if somehow the source or a sample set of generated documents from the same source can be obtained, it is possible to detect generated documents in a mixture corpus

	False positive rate	False negative rate
Markov model	0.28	0.04
RNN	0.0	0.08

Table 3: False positive and false negative rate for SciDetect with Markov model and RNN.

of different text types. Even though these methods of text generation can be improved by increasing the number of training sample, we believe that the overall result will not be much different where generated texts would be "centered" around the model that was learned on the topic.

However, even if texts generated by these methods are somewhat understandable, they are not guaranteed to have a proper spelling nor proper grammatical structure. Furthermore, it is also hard to replicate more complicated elements such as tables or figures. Hence, we decided to focus only on how to detect texts generated by PCFG.

It has been shown that there are several methods to detect text that was generated by PCFG. Nevertheless, all of them are working at the document level and are unable to detect a small quantity of generated text inside a large body of genuinely written text, thus making them easier to be deceived. That was the reason the parse trees are investigated as a mean to determine the similarity between sentences.

4 Measure Sentence Similarity with Parse Tree and The Use of Grammatical Structure Similarity

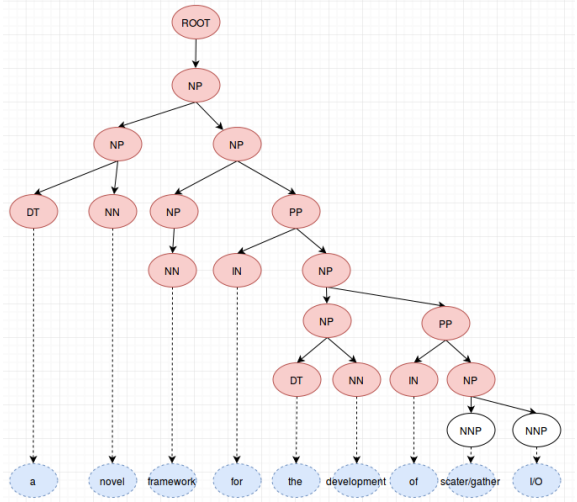
This section shows different approaches using a parse tree to quantify the similarity between sentences. Furthermore, we define a former definition to measure sentences similarity based on their structures.

4.1 Using Parse Tree to Measure Sentence Similarity

A parse tree is a tree that represents the syntactic structure of a sentence or a phrase (Example 41 and 42). Each sentence can be separated into Verb Phrase (VP), Noun Phrase (NP) and then deeper level as Noun, Verb, Adjective, etc.... This might make one think that sentences with a similar structure and word pairs might be related to each other.

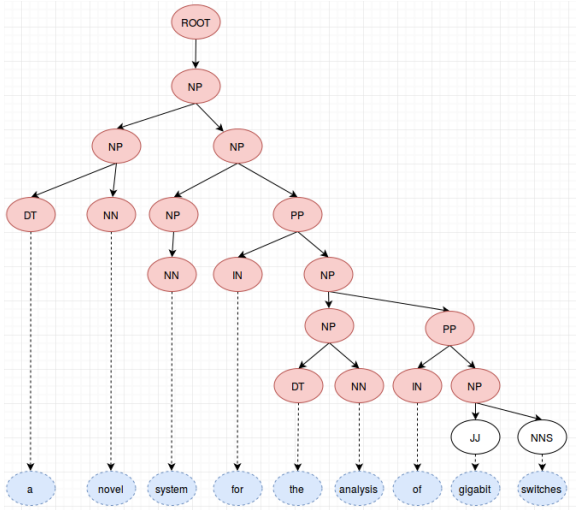
Example 41. A parse tree in different forms for the phrase: "a novel framework for the development of scater/-gather I/O".

(ROOT(NP(NP(DT a) (NN novel)) (NP(NP(NN system)) (PP(IN for) (NP(NP(DT the) (NN analysis)) (PP(IN of) (NP(NNP scater/gather) (NNP I/O))))))))



Example 42. A parse tree in different forms for the phrase: "a novel system for the analysis of gigabit switches".

(ROOT(NP(NP(DT a) (NN novel)) (NP(NP(NN system)) (PP(IN for) (NP(NP(DT the) (NN analysis)) (PP(IN of) (NP(JJ gigabit) (NNS switches))))))))



Over the years, there have been multiple proposals using a parse/dependency tree to discover the similarity between sentences. For example [19] uses their sentence similarity measurement in Exactus [15] to detect plagiarism. This measurement uses different characteristics from the sentence including TF-IDF, IDF overlap and a syntactic similarity measurement where the syntactic links between pairs of words from different sentences are measured. Using this method, they could obtain the second highest score for the plagiarism detection track at the PAN workshop in 2014 [6].

[4] proposes a method to estimate the similarity between sentences using the number of common tree segments in their augmented parse tree. This augmented parse tree is created using a normal parse tree structure, but each node is represented as a feature vector instead of an entity in the sentence. The similarity score between two augmented trees is then defined using the tree kernels function which uses both the matching subsequence of the children of each node and the compatibility between two feature vectors.

The DLSITE-2 system [16] aims at determining the truth of a text fragment from another text (textual entailment) using a syntactic parse tree. In this work, sentences are selected based on words with significant grammatical value likes nouns, verbs, adjectives, etc. Sentences with the same or similar significant grammatically value words are parsed to syntactic trees and these trees are compared to detect if one is contained by another. This information is then used to demonstrate that it is possible to deduce textual entailment from a hypothesis using syntactic trees.

Parse tree, along with common words, are also used by [5] to determine the similarity between sentences. They propose a method to search for semantic relation based on the exploding of a parse tree starting from pairs of similar words. For each pair of sentences, the syntactic dependency trees

[6] http://pan.webis.de/clef14/pan14-web/

are obtained using Stanford Parser[7], then the most significant terms of each sentence, like nouns or verbs, are discovered. From those terms, the tree is explored by going up to the ancestors step by step, until a connection is formed; this connection is used as a common subtree between the two trees. The similarity between them is then calculated with the number of common nodes along with custom weight for each tree.

However, these methods might not fit our particular need since they either focus on pairs of common word or are too expensive when it is required to parse every single sentence. Thus section 4.2 shows our framework to detect automatically generated sentences.

4.2 Grammatical Structure Similarity

This section shows how the data is handled and the definition of grammatical structure similarity.

For pdf documents in the test corpus, first they are converted to plain text, then normalized (de-capitalize, remove numbers, symbols, non-conventional characters, etc.). Later these texts are separated into sentences, and each sentence is parsed using Stanford Parser[7] to obtain a parse tree. Since in our case, the keywords have little to no value in deciding the similarity between sentences. They are removed from the structure, and only the nodes are kept (Example 43). These parse trees are then compared to the PCFG corpus of known parse trees from pre-processing generated sentences using a recursive loop to find the biggest possible subtree match of the tree structure.

Example 43. The parse tree in example 41 would be considered only as:
(ROOT(NP(NP(DT) (NN)) (NP(NP(NN)) (PP(IN) (NP(NP(DT) (NN)) (PP (IN) (NP(NNP) (NNP))))))))

Once a similar structure is found, the similarity between them need to be quantified. So, the **grammatical structure similarity** is defined as follows:

Definition 1. *Grammatical structure similarity (GSS):*
Let N_A be the number of nodes in the parse tree T_A of sentence A, N_B be the number of nodes in the parse tree T_B of sentence B, and N_{AB} be the number of nodes in the biggest common subtree of T_A and T_B. Then the Grammatical Structure Similarity between A and B is defined as:

$$GSS_{(A,B)} = \frac{2*N_{AB}}{N_A+N_B}$$

Example 44. Grammatical Structure Similarity between Example 41 and Example 42

$$GSS_{(E_{21}/E_{22})} = \frac{2*17}{19+19} = 0.89$$

In our proposal, the computation is quite expensive since every sentence needs to go through the parser and then is compared with all samples in the PCFG corpus. This will be explored further in the following section.

5 GSS System with Jaccard Filter and Sentence's Context

This section presents the process to build a complete system to detect automatically generated sentences. At first multiple corpora of generated documents are presented and then tested to determine the most appropriate one for our needs. Then, to reduce the processing time, a filter is implemented to cut down the number of sentences that need to be parsed. Then in Section 5.4, a fully developed system to detect automatically generated sentences is presented.

5.1 Corpora

Multiple text corpora are used for latter testing purpose and this section will give a detailed description about them.

PCFG Corpus: PCFG corpora of different sizes were used as samples and tried to learn the different parse tree structures from the generators. Table 4 shows the correlation between the size of the particular corpus (evenly distributed between four generators) with the number of sentence and the number of distinct parse tree that can be obtained from those sentences.

Even though the number of distinct sentences and trees increase steadily, it is possible that there are only small variations between them because they were generated using the same rule as seen in example 41 and 42. This will be tested in section 5.2 later.

Test Corpus: This corpus is composed of 4 smaller corpora each containing 100 texts from an automatic generator and a real corpus of 100 genuine human written texts which were selected at random from different fields. This resulted in about 110k sentences being used as the test corpus.

Corpus size	nb of sentences	nb of distinct sentences	nb of distinct parse trees
80	12.1k	9.2k	8k
160	25.5k	18.2k	14.8k
320	45.5k	33.2k	26.9k

Table 4: Number of sentences, distinct sentences and parse trees for different corpus size

5.2 GSS and PCFG Corpus Effectiveness

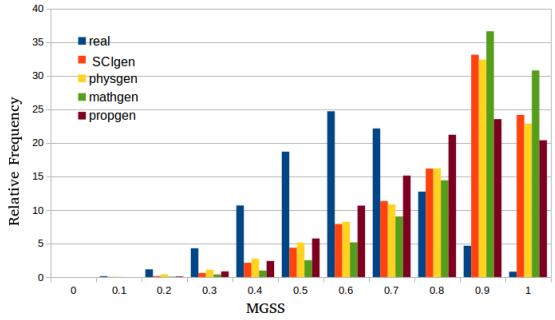
Our hypothesis is that even though the number of distinct sentences as well as parse trees seem quite numerous(4), most of them should also be somewhat similar to each other. Only a small proportion of the sentences are different. To verify this, the maximum GSS (MGSS) of all sentences in the test corpus for three different size PCFG corpora that were computed and presented in section 2 and the results are shown in Figure 3.

Definition 2. *Maximum Grammatical Structure Similarity (MGSS): For a sentence A in the test corpus (C_T), The MGSS between A and the PCFG corpus (C_{PCFG}) is:*

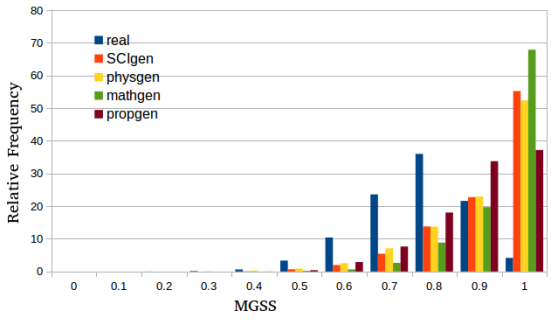
$$MGSS_{(A,C_{PCFG})} = Max_{(B \in C_{PCFG})}(GSS_{A,B})$$

It is understandable when comparing the PCFG corpus of size 80 with the others. With more samples in the PCFG corpus, it is possible to find much higher GSS matches for generated sentences. However, comparing the PCFG of size 160 and 320, it is difficult to see any significant difference. This suggests that the previous hypothesis is true. Even though the size of the PCFG sample corpus was doubled, it did not double the match rate because most of the additional parse tree structures have very little differences with what has already been obtained in the smaller size corpus. Subsequently, from now on, only the PCFG corpus of size 160 is used.

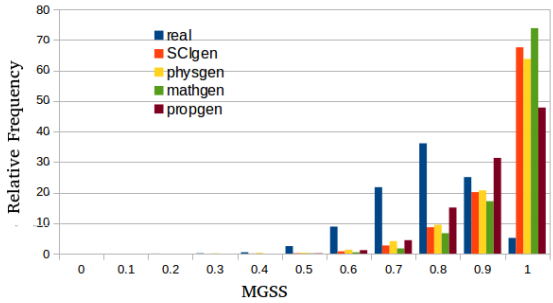
Figure 3 also shows that for SCIgen, physgen and mathgen it is possible to find more than 50% of really high matches (GSS higher than 0.9) as compared to less than 2% for genuinely written sentences. Even though there is no clear separation for the score, it is easy to see that there are



PCFG corpus with 80 samples



PCFG corpus with 160 samples



PCFG corpus with 320 samples

Fig. 3: The distribution of MGSS for sentences in the test corpus with the PCFG corpora.

different bell curves for genuinely written and generated ones. The curves for generated sentences lean very heavily toward the end of the histogram thus making them stand out.

Furthermore, Table 5 shows some examples of genuinely written sentences with high GSS to other sentences in the PCFG corpus where Jaccard similarity is the number of common words over the total number of distinct words in two sentences. It can be seen that most of them are just common sentences that also appear in the PCFG corpus. To deal with such problems, the context of the sentence is taken into account, and this will be presented in section 5 .

5.3 Sentence Filter Using Jaccard similarity

As mentioned before, the cost for parsing is quite expensive, so this raises the need to implement a filter to reduce the number of sentences that need to be parsed. To do such a task, the Jaccard similarity was used. Figure 4 shows the Jaccard similarity between sentences in the test corpus with the sentences in the PCFG corpus that have the highest GSS to them.

As shown in Figure 4, the majority (more than 90%) of genuinely written sentence have Jaccard similarity less than 0.3 to sentences in the PCFG corpus, while it was only about 20% for other types of generator (except about 40% for propgen). Subsequently, this would make 0.3 a good candidate

Genuine written sentence	Sentence in PCFG corpus	Jaccard similarity	GSS
our main contributions are as follows	our main contributions are as follows	1	1
it is easy to see that	it is easy to see that	1	1
the states of this network are	the contributions of this work are as follows	0.4	0.89
the rest of the paper is organized as follows	the rest of this paper is organized as follows	0.88	1
the remainder of the paper is organized as follows	the rest of this paper is organized as follows	0.8	1
the proof of the claim can be found in appendix	useful survey of the subject can be found in	0.58	0.9
the interpretation of the walk is as follows	the rest of the paper proceeds as follows	0.45	1

Table 5: Some typical mistakes from using only GSS

for a threshold to be used in the filter since it is possible to keep many "suspected generated" sentences while greatly reducing the number of "irrelevant" sentences.

Even though Jaccard similarity can filter around 90% of the sentences, at the same time 20% of genuinely written sentences were also marked. This would result in a large number of false positives. However, this filter significant reduces the computational cost since it is no longer required to parse and compare each sentence within the PCFG corpus, only those that are similar to a generated sentence.

5.4 Fully Developed GSS system

Since the aim is to detect a small portion of automatically generated text, each sentence is considered along with its context, which includes the direct previous and next sentence to balance out special cases. Furthermore, in our preliminary test, sentences with less than 5 words provide little information while sentences with more than 35 words might cause trouble with the parser. Thus, for each sentence in the test that is longer than 5 words and less than 35 words, the PCFG corpus is used to find other sentences that have Jaccard similarity higher than 0.3. Then the GSS between them are calculated to obtain the maximum result; the same process is repeated for the previous and the next sentences. The final GSS with context for the sentence is the average GSS of itself along with its direct neighbors.

The result for the Jaccard filter is shown in Table 6. It can be seen that the filter seems to serve its purpose. Even though on average there are more sentences in a genuinely written paper, only 20% of them pass the filter and need to be parsed as compared to 70% to more than 90% of sentences in automatically generated papers. This greatly reduced the processing time required,

since, in reality, one would assume that an overwhelming number of sentences are genuinely written.

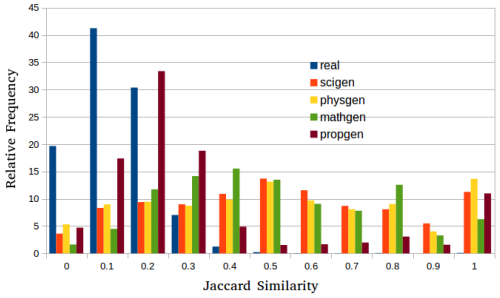


Fig. 4: Relative frequency of maximum Jaccard similarity between different type of sentences to the PCFG corpus

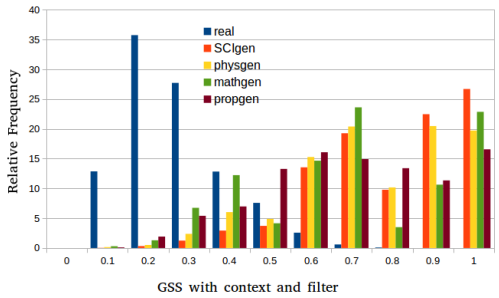


Fig. 5: Relative frequency of GSS with context and Jaccard filter

	Real	SCIgen	Physgen	Mathgen	Propgen
Avg. number of sentence in a paper	192.4	87.2	81.7	174.0	88.9
Avg. number of sentence that need to be parsed	38.8	80.0	73.8	161.4	62.9
Avg. percentage of sentence that need to be parsed	20.2%	91.7%	90.2%	92.7%	70.7%

Table 6: Average number and percentage of sentence in a paper compare to sentence that need to be parsed.

The result of the GSS system using the Jaccard filter and GSS with context is shown in Figure 5. It shows that a majority (96.7%) of genuinely written sentences which pass through the filter have less than 0.5 GSS. On the other hand, for automatically generated sets of sentences if a threshold is set at 0.5, it is possible to detect more than 90% for SCIgen, physgen and about 75% mathgen and propgen.

5.5 Complexity of the System and Average Processing Time

It is understandable that the system is quite complex. There are several places that can be evaluated for complexity such as:

- Compare each sentence with all other sentences in the PCFG Corpus using Jaccard similarity as a filter. Complexity $O(1)$
- If the sentence get past the filter, it need to be parsed by the Stanford Parser. This is the costliest step with complexity $O(n^3)$
- For each pair of sentences that go through the filter together, their parsed trees are compared to find the biggest common subtree. This cost $O(n^2)$

Eventually the overall complexity of the system is $O(n^3)$. In practice, the run time for each document in our system [7] depends on the type of document. However, typically a fully genuinely

[7] Intel Core I5 2.4 GHz with 16Gb Ram

written document would pass through every 10 to 20 seconds while a fully generated one would take much longer from 60 to more than 90 seconds. This will not affect a real -life work flow much since the majority of documents that need to be scanned are presumably all genuinely written.

Nevertheless, it is still necessary to thoroughly evaluate the performance of the system against some other techniques. This will be addressed in the next section.

6 Comparison with Other Methods and Evaluation

To effectively evaluate the performance of the system, some methods to classify sentences are used. This includes a modified pattern checker and some typical machine learning techniques.

6.1 Pattern Checker

We were able to obtain the Pattern Checker program from Springer that was presented earlier in Section 3 and modified it to discover the pure number of detected patterns for each sentence. The modified program scans the document and marks all the suspected patterns, then the sentences that contain those detected patterns are marked as automatically generated regardless of their length or structure. The result is shown in Table 7; since the program only focuses on SCIGen patterns so the test was only performed on SCIGen and SCIGen-Physic samples.

	False positive rate	False negative rate
SCIGen	0.0	0.68
SCIGen-Physic	0.0	0.86
Real	0.12	0.0

Table 7: False positive and false negative rate for Pattern checker method.

Table 7 shows that a simple pattern checker method still misses many generated sentences. However, in using in unison with the scoring method, this method can detect fully generated documents. Nevertheless, this method could not detect many of generated sentences and would be impractical in the case of a small, partially generated document or a modified generator.

6.2 Traditional Machine Learning Techniques

To further evaluate the approach, R and Rtexttools package[3] are used; this package is used for supervised learning and includes different learning algorithms. The PCFG corpora were converted to texts and separated into sentences. Since it is also required to have samples for genuinely written sentences, other real corpora of the same size to the counterpart PCFG corpora were chosen at random to be used as genuine-sample-corpora.

The test corpus was split into sentences, they are classified with different methods using a document-term-matrix with remove spare terms at 0.98 to obtain each one a label as “generated” or “real”. In the end, about 57.5k automatically generated sentences and 118.5k genuine written sentences were used trying to classify other 19k generated sentences beside 43k genuinely sentences.

It is understandable that for a truly impartial comparison, the information from the parse trees should also be given to the classifiers. However, transforming a parse tree to a feature vector is not a straightforward task and it may even be impossible in practice. If trees would be added as a particular feature for learning algorithms, then one would have to define how to compute a similarity for this is particular feature and this exactly what GSS is defining.

Considering precision or recall can be easily manipulated by using different size corpora. For example, if a test corpus is composed of ten thousand automatically generated sentences and only ten of genuinely written ones and if a classification method just marked everything as automatically generated. Then the Precision which is the division of true positive over the sum of true positive with false positive, as well as Recall, which is the division of true positive over the sum of true positive with false negative are both very high even though all the genuinely written document are marked as automatically generated.

However, a false positive rate which is the probability of a genuinely written sentence marked as generated and vice versa for false negative rate are independence from the corpus size, so they are used to fairly represent the results. Table 8 shows that conventional machine learning methods might not be appropriate to our need since the results vary from very bad (Glmnet, SLDA, Tree) where most of the sentence were marked as "generated" to mediocre (Max entropy, boosting, bagging random forest) where they marked about half of the genuine written sentences as "generated". For the GSS only and GSS system, 0.5 was used as a threshold to determine whether a sentence is automatically generated. As seen in Figure 3 with GSS only, this is not a good threshold for a single sentence however, if the context is taken into account as in GSS system (Figure 5), a very promising result are obtained with very few genuine sentences marked as "generated" (for corpus size 160 there were less than 200 sentences marked as automatically generated out of more than 22k genuine written sentences) but still catch a good number of automatically generated ones.

Furthermore, to verify the possibility of detecting a modified generator where only the terminal terms or keywords were changed. Physgen, which is only a version of SCIgen with all the "hot keywords" switched from computer science to physic ones, is used.

For this test, a corpus of 40 SCIgen papers is used to try to detect physgen sentences in 100 physgen papers and 100 genuine papers. As before, a genuine-sample-corpus of 40 genuine papers is also used to aid machine learning techniques. The results are shown in the "40 SCIgen" column of Table 8. As suspected, using GSS only could catches most of the sentences from physgen with only samples from SCIgen (0.015 false negative rate); even if the context and Jaccard filter are used, GSS system is still able to find 80% of them. This suggests that the GSS system would also be effective against cases of newly modified versions of existing generators.

	False positive rate				False Negative rate			
corpus size / algorithm	80	160	320	40 SCIgen	80	160	320	40 SCIgen
Glmnet	0.03	0.02	0.04	0.03	0.80	0.87	0.80	0.63
Maxentropy	0.19	0.13	0.20	0.14	0.55	0.62	0.52	0.45
SLDA	0.01	0.01	0.05	0.04	0.95	0.97	0.78	0.63
Boosting	0.58	0.5	0.62	0.14	0.23	0.11	0.21	0.49
Bagging	0.10	0.05	0.1	0.06	0.37	0.5	0.35	0.42
Random Forest	0.07	0.05	0.07	0.05	0.49	0.58	0.43	0.44
Tree	0.02	0.02	0.02	0.03	0.87	0.88	0.87	0.65
GSS only	0.85	0.95	0.97	0.73	0.07	0.007	0.002	0.015
GSS system	0.008	0.008	0.01	0.002	0.25	0.19	0.17	0.21

Table 8: False positive and false negative rate of different methods with different corpus size

6.3 Evaluation

To accurately evaluate the performance of the system, a small evaluation is performed as follows. 10 genuine documents are used, and each is injected with 50 different sentences from SCIgen in two random paragraphs of 25 sentences each as seen in Figure 6.

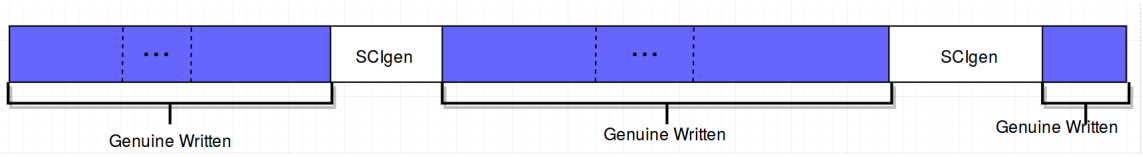


Fig. 6: Example structure of a genuine written document injected with automatically generated sentences

These documents are then run through the GSS system to observe how many automatically generated sentences would be picked up in this situation. The result of this test is quite encouraging since for each document, on average 33.57 out of 50 automatically generated sentences are detected. In the worst case, the system could detect 27 sentences, and closer examination of this case reveals that one of the SCIgen paragraphs contains some short, un-structured mathematical formulas. This one again confirms our assumption that the system can detect small paragraphs of automatically generated text inside a genuinely written document.

7 Conclusion

There is a need for automatic detection of automatically generated texts to ensure a certain level of quality for bibliography service. In this paper, various methods to generate text have been presented and tackled. Particularly, documents that were generated by a PCFG, a RNN or a Markov model have all been classified with reasonably good results. Still, PCFG seems to have been used the most and is focused on by many other researches.

However, current detection approaches for PCFG all focus on the document level. So, in this paper we have shown our GSS system which can detect sentences from known PCFG generators with sufficient samples, which has an 80% positive detection rate and less than a 1% false detection rate. Furthermore, the system has been tested against some well-known machine learning techniques to demonstrate that it can provide the best results. The possibility of detecting a modified version of current generators was also verified with great success.

However, using the system against generators which use other techniques, such as Markov model or RNN is impractical. Thus, textual distance was used to show that generated texts from a Markov model or a RNN still share a somewhat similar distribution to the original source. If somehow the source or other samples of generated texts from the same source can be found, it is possible to classify them with a certain probability.

Despite all of that, the system is still in need of many improvements. One obvious change would be the implementation of a lemmatization before the parsing step, as this would promise a better result. Otherwise, it could also be improved on to work as a plagiarism detection where words are changed from the original sentences.Nevertheless, current results are still quite promising for the desired goal.

References

1. Amancio, D.R.: Comparing the topological properties of real and artificially generated scientific manuscripts. Scientometrics 105(3), 1763–1779 (Dec 2015)
2. Bohannon, J.: Who's afraid of peer review? Science 342(6154), 60–5 (Oct 2013)
3. Collingwood, L., Jurka, T., Boydstun, A., Grossman, E., van Atteveldt, W.: Rtexttools: A supervised learning package for text classification. The R Journal 5(1), 6–13 (2013)
4. Culotta, A., Sorensen, J.: Dependency tree kernels for relation extraction. In: Proceedings of the 42Nd Annual Meeting on Association for Computational Linguistics. ACL '04, Association for Computational Linguistics, Stroudsburg, PA, USA (2004)
5. Durán, K., Rodríguez, J., Bravo, M.: Similarity of sentences through comparison of syntactic trees with pairs of similar words. In: Electrical Engineering, Computing Science and Automatic Control (CCE), 2014 11th International Conference on. pp. 1–6 (Sept 2014)
6. Ginsparg, P.: Automated screening: ArXiv screens spot fake papers Nature - 508(- 7494), 44 (Mar 2014)
7. Klein, D., Manning, C.D.: Fast exact inference with a factored model for natural language parsing. In: In Advances in Neural Information Processing Systems 15 (NIPS. pp. 3–10. MIT Press (2003)
8. Labbe, C.: Ike Antkare one of the great stars in the scientific firmament. ISSI Newsletter 6(2), 48–52 (2010)
9. Labbé, C., Labbé, D.: Duplicate and fake publications in the scientific literature: How many scigen papers in computer science? Scientometrics 94(1), 379–396 (Jan 2013)
10. Labbé, C., Labbé, D., Portet, F.: Detection of Computer-Generated Papers in Scientific Literature, pp. 123–141. Springer International Publishing (2016)
11. Lavoie, A., Krishnamoorthy, M.: Algorithmic detection of computer generated text. arXiv preprint arXiv:1008.0706 (2010)
12. López-Cózar, E.D., Robinson-Garcia, N., Torres-Salinas, D.: Manipulating google scholar citations and google scholar metrics: simple, easy and tempting. CoRR abs/1212.0638 (2012)
13. Nguyen, M., Labbé, C.: Engineering a tool to detect automatically generated papers. In: Proceedings of the Third Workshop on Bibliometric-enhanced Information Retrieval co-located with the 38th European Conference on Information Retrieval (ECIR 2016). pp. 54–62 (2016)
14. Noorden, R.V.: Publishers withdraw more than 120 gibberish papers. Nature News (Feb 2014)
15. Sochenkov, I., Zubarev, D., Tikhomirov, I., Smirnov, I., Shelmanov, A., Suvorov, R., Osipov, G.: Exactus like: Plagiarism detection in scientific texts. In: European Conference on Information Retrieval. pp. 837–840 (2016)
16. Wang, R., Neumann, G.: Recognizing textual entailment using sentence similarity based on dependency tree skeletons. In: Proceedings of the ACL-PASCAL Workshop on Textual Entailment and Paraphrasing. pp. 36–41. RTE '07, Association for Computational Linguistics, Stroudsburg, PA, USA (2007)
17. Williams, K., Giles, C.L.: On the use of similarity search to detect fake scientific papers. In: Similarity Search and Applications - 8th International Conference, SISAP 2015. pp. 332–338 (2015)
18. Xiong, J., Huang, T.: An effective method to identify machine automatically generated paper. In: Knowledge Engineering and Software Engineering. pp. 101–102 (2009)
19. Zubarev, D., Sochenkov, I.: Using sentence similarity measure for plagiarism source retrieval. In: CLEF (Working Notes). pp. 1027–1034 (2014)