IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Vol. 10, Issue 11, November 2021

DOI: 10.17148/IJARCCE.2021.101131

Graph-Based Comma Splice Correction Using Sentence Dependency Parsing

Subham Sahu[1], Jeevanlal Kori[2], Jitendra Singh Thakur[3]

Department of Computer Science and Engineering, Jabalpur Engineering College, India[1,2,3]

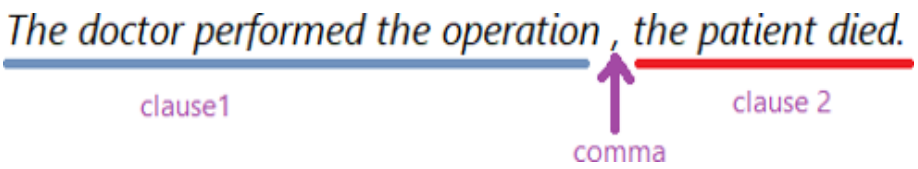
Abstract: Grammatical Error Correction (GEC) systems aim to automatically fix errors in text written by language learners. Most current GEC tools focus on word-level and phrase-level mistakes, such as spelling, preposition, and article errors. However, sentence-level errors are also common in learner writing. In this paper, we focus on correcting a specific sentence-level error: the comma splice. Our approach uses sentence dependency parses to detect and correct these errors. On our evaluation dataset, our method achieved 53.70% precision and 28.43% recall.

1. INTRODUCTION

English is widely used around the world, but non-native speakers often struggle to master its grammar rules. Violating these rules leads to grammatical errors, which are particularly noticeable in written text. Consequently, Grammatical Error Correction (GEC) has become an important research field. Several benchmark shared tasks have driven GEC progress, including works by Ng et al. [7] (CoNLL 2014), Dahlmeier et al. [2] (NUCLE), and Bryant et al. [1] (BEA-2019). Most participating systems target local errors like wrong prepositions or noun number mismatches, while neglecting sentence-level issues such as run-ons and sentence fragments [9].

A comma splice is a sentence-level error that occurs when two independent clauses are joined with only a comma. See figure 1. To fix a comma splice, one can insert a coordinating conjunction after the comma ("The doctor performed the operation, and the patient died.") or replace the comma with a period and capitalize the next word ("The doctor performed the operation. The patient died.").

Figure 1: Comma Spice example



To correct the comma splice error, either we can add a proper conjunction between two clauses (“The doctor performed the operation and the patient died.”) or we can replace the comma with a period sign and capitalize the first letter after it (“The doctor performed the operation. The patient died.”). In literature very few efforts have been made for correcting comma splice errors. Israel et al. [3] correcting comma errors (excluded comma splice error) as sequence labelling task using CRF model. Lee et al. [5] also approached correcting comma errors as a sequence labelling task using CRF model, and primarily focused on correcting comma splice errors. Tilk et al11] proposed a punctuation restoration system, which is useful in restoring punctuations in ASR (Automated Speech recognition) systems, because most ASR systems output unpunctuated text. Thereafter, Zheng et al. [12] focused on correcting run on errors (excluded comma splices).

In this work, we propose a rule-based approach using the Stanford Dependency Parser to detect and repair comma splices. If a sentence violates structural graph rules, it is flagged as containing a comma splice, and the comma is replaced with a period. The rest of this paper is structured as follows: Section 2 reviews related literature. Section 3 details our proposed system. Section 4 presents experimental setup and results. Section 5 discusses current limitations, and Section 6 provides conclusions and future work.

2. LITERATURE REVIEW

Israel et al. [5] presented one approach to correct comma errors (excluded comma splice) in Learners text. They approached problem of correcting commas as a sequence labelling problem. Conditional Random Field (CRF) is

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Vol. 10, Issue 11, November 2021

DOI: 10.17148/IJARCCE.2021.101131

used to label every space between words of the sentence, with comma or no comma. He used features like n-gram and POS and distance features to train CRF model. For training they artificially created the dataset. His average performance on two corpora (Learners essay and News wire text) is 89% precision and 25%.

The most closely related to our work is Lee et al. [5]. This paper is primarily focused on correcting comma splices errors. He approached correcting comma splice error as sequence labelling task, each comma in sentence is tagged by CRF model as True or False. In addition to features like n-gram and POS, he used some additional features like clause features and parse features (patterns in parse tree). For training they also artificially generated the data, example sentences are taken from Penn Treebank. On testing, system achieved average of 0.91 precision and 0.28 recall, over two datasets viz. CityU Set and Cambridge Set.

Zheng et al. [12] also used sequence labelling, but for correcting run-on errors (excluded comma splice). They proposed two models for this task, Conditional random fields (roCRF) and Sequence to sequence model (roS2S). In this roCRF model, labels each word with SPACE or PERIOD. And roS2S model works as neural sequence generation and it's based on a neural machine translation model Sutskever et al.[10]. For training of both models, they have artificially generated data from corpus of Napoles et al. [6] clean newswire text, Annotated Gigaword. For testing use NUCLE corpora [2] CoNLL shared task [7]. Finally, roS2S has highest performance with $F_{0.5}$ score 0.86 and roCRF has very highest precision P between 0.83− 0.89, but roCRF has less recall. Thereafter, Tilk et al [11] used a bidirectional recurrent neural network model with attention mechanism for punctuation restoration (including commas) in unsegmented text. Their two best models T-BRNN (without pre-trained embeddings) and T-BRNN-pre (with pre-trained embeddings) achieved 68.9 precision, 58.1 recall and 63.1 $F_{0.5}$ score. This research is useful in restoring punctuations in ASR systems.

3. PROPOSED METHODOLOGY

3.1 Dependency Parse approach

3.1.1 Comma Splice Detection Graph (CSD)

For detection of comma splice error, we have used dependency parse of the sentence (obtained by Stanford Dependency Parser). We have analysed dependency parse for sentences having comma splice errors. After doing analyzing, we have formulated a method to detect comma splice. First, we obtain the dependency parse of the sentence, then using this dependency parse, we construct a graph, called CSD (comma splice detection) graph. CSD graph will tell us whether a sentence have comma splice or not. So, now let us define all the steps required to construct CSD graph, for a given sentence.

(a) Step 1: Parse the given sentence using Stanford dependency parser.
(b) Step 2: Find nodes for CSD graph. First node is always found at head of the root edge, after which we traverse down the dependency parse, and find all those nodes, which are connected to first node via a series of **ccomp** or **conj** dependency edges. (from our analysis we have found that these nodes are generally verbs in a sentence, for example if a sentence have three clauses, then we may have at max three nodes in the CSD graph).
(c) Step 3 : Connect CSD nodes.

For each CSD node, we perform following sub-steps.

(a) Let X be our current CSD node, now check from X, is there any **cc** edge (generally found in compound sentences) or **mark** edge, (generally found in complex sentences). Ensure one more thing that if it is a 'cc' edge, then it's head should point toward a coordinating conjunction ('and', 'but' etc.) and if it is a 'mark' edge, then it's head should point toward a subordinating conjunction ('that', 'if' etc.). If X satisfies all conditions, then check if X is connected with root node (first node) using 'conj' edge or 'ccomp' edge respectively in dependency graph, if so then connect X and first node in CSD graph also.

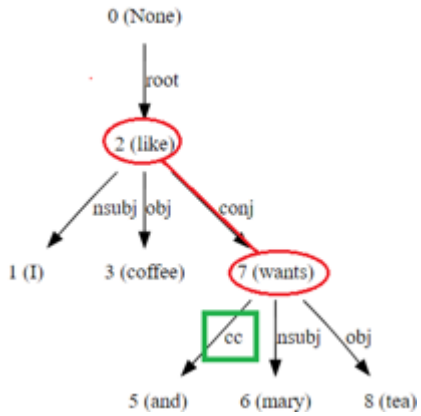
IJARCCE

ISSN (O) 2278-1021, ISSN (P) 2319-5940

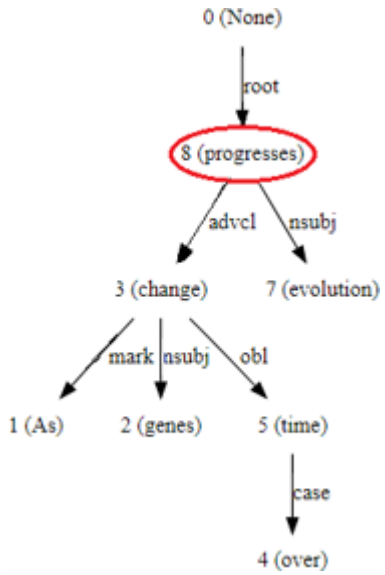
International Journal of Advanced Research in Computer and Communication Engineering

Vol. 10, Issue 11, November 2021

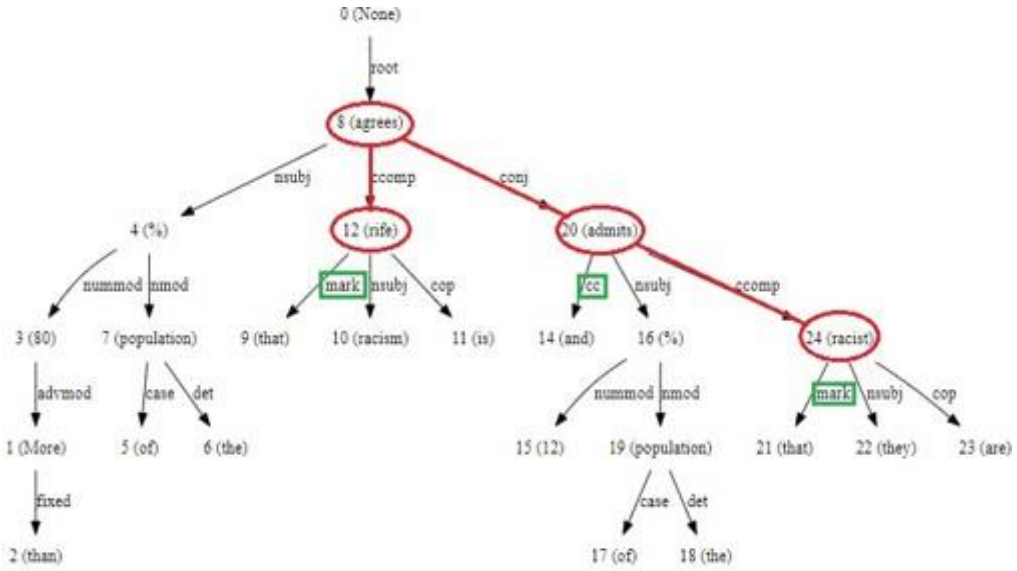
DOI: 10.17148/IJARCCE.2021.101131



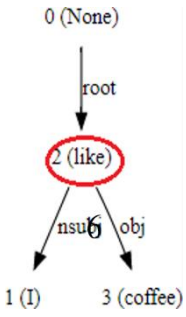
(a) Compound sentence



(b) Complex sentence



(c) Compound Complex Sentence with 4 clauses



(d) Simple sentence

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Vol. 10, Issue 11, November 2021

DOI: 10.17148/IJARCCE.2021.101131

(b) For all such Xs, which satisfies all conditions of step 2(a), but not connected with first node, connect them with some CSD node which is already connected to first node in CSD graph, using 'conj' dependency edge or 'ccomp' dependency edge.

Example 1: "I like coffee, but marry wants tea.". See it's dependency parse in figure 2 (a), obtained using step 1. From step 2, we have found two nodes for CSD graph (RED Circles). ('wants' and 'like' are only main verbs of the sentence). Now from step 3, we have connected these nodes (RED straight line) via a 'conj' dependency edge (GREEN box represents founded 'cc' edge in step 3).

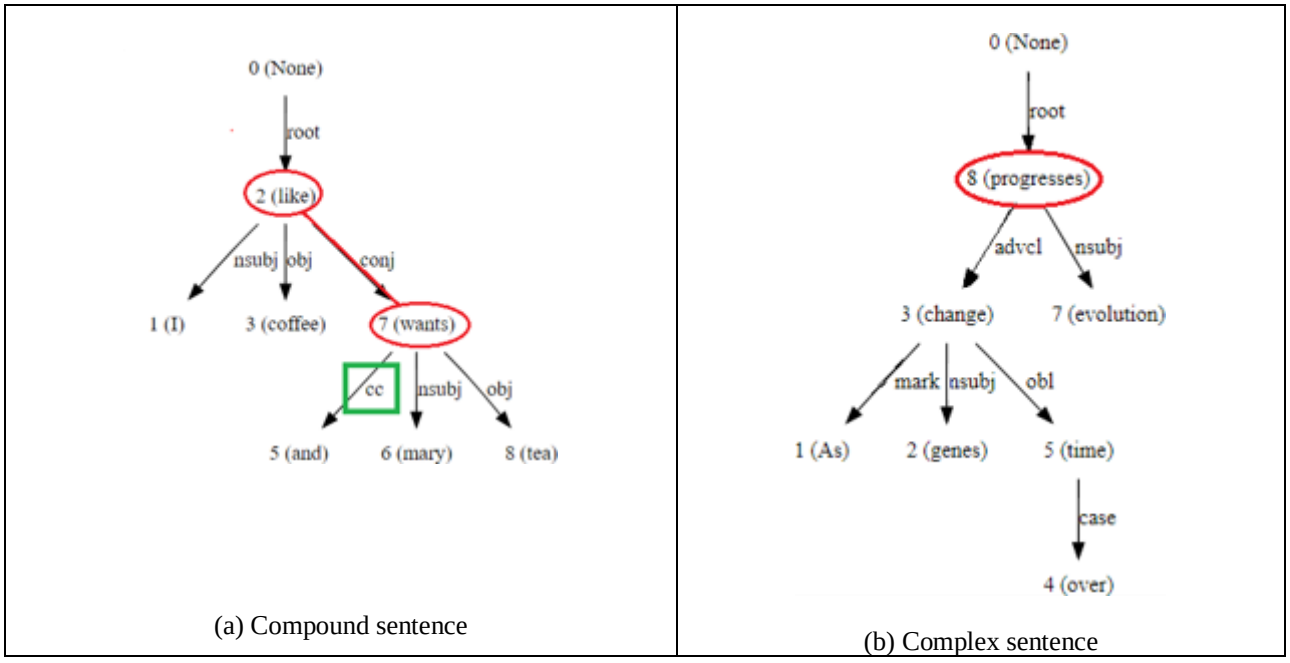
Example 2: "As genes change over time, evolution progresses.". See it's dependency parse in figure 2 (b), obtained using step 1. From step 2, we have found only one node for CSD graph ('progress' is only main verb in the sentence). Since, we have only one node in the CSD graph thus, no need to perform step 3.

Example 3: "More than 80% of the population agrees that racism is rife, and 12% of the population admits that they are racist." See it's dependency parse in figure 2(c), obtained using step 1. From step 2, we have found four nodes for CSD graph. Now from step 3, we have connected these nodes via **'ccomp'** and **'conj'** edges (GREEN box represents founded 'cc' and 'mark' edges in step 3).

Example 4: "I like coffee.". See it's dependency parse in figure 2(d), obtained using step 1. From step 2, we have only one node for CSD graph ('like' is only main verb in the sentence). Here also, no need to perform step 3.

How to detect comma splice error using CSD graph? If the resultant CSD graph is not connected, then we will conclude that two or more clauses are improperly joined with a comma between them, and we say sentence has comma splice error. Too understand it better consider, if we remove the coordinating conjunction 'and' from example 1, 3 and subordinating conjunction 'as' from example 2 and reconstruct there CSD graphs, then one can clearly see in figure 3 that CSD graphs are not connected.

Figure 2: Dependency parse and CSD graph for correct sentence



(a) Compound sentence

(b) Complex sentence

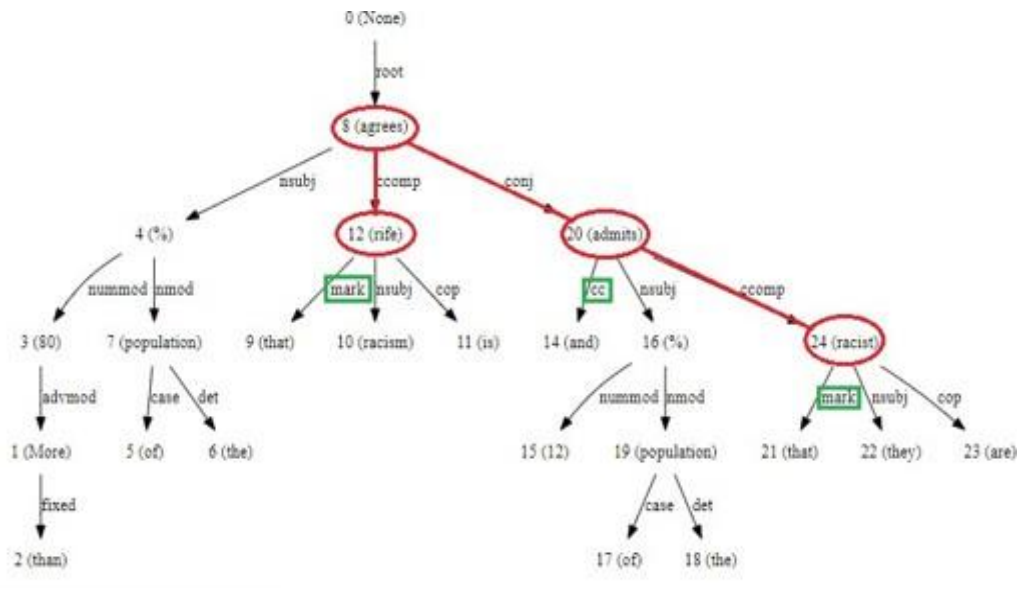
IJARCCE

ISSN (O) 2278-1021, ISSN (P) 2319-5940

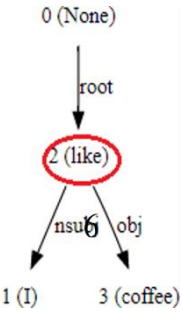
International Journal of Advanced Research in Computer and Communication Engineering

Vol. 10, Issue 11, November 2021

DOI: 10.17148/IJARCCE.2021.101131

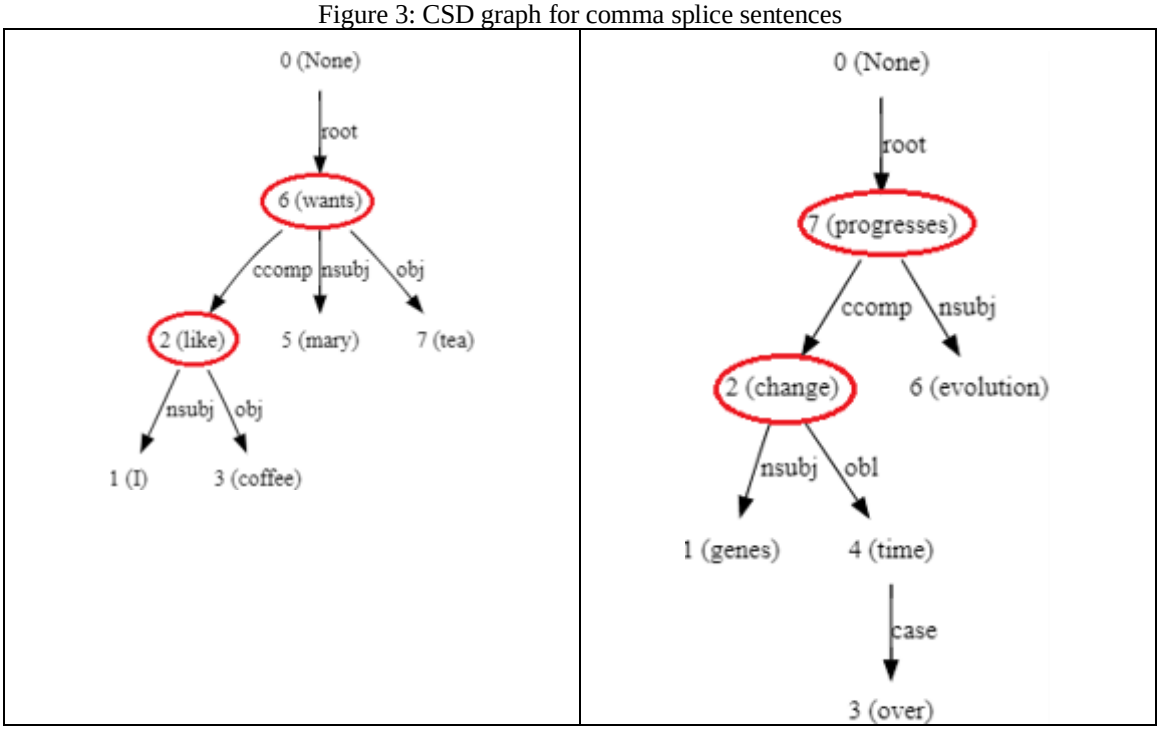


(c) Compound Complex Sentence with 4 clauses



(d) Simple sentence

Figure 3: CSD graph for comma splice sentences



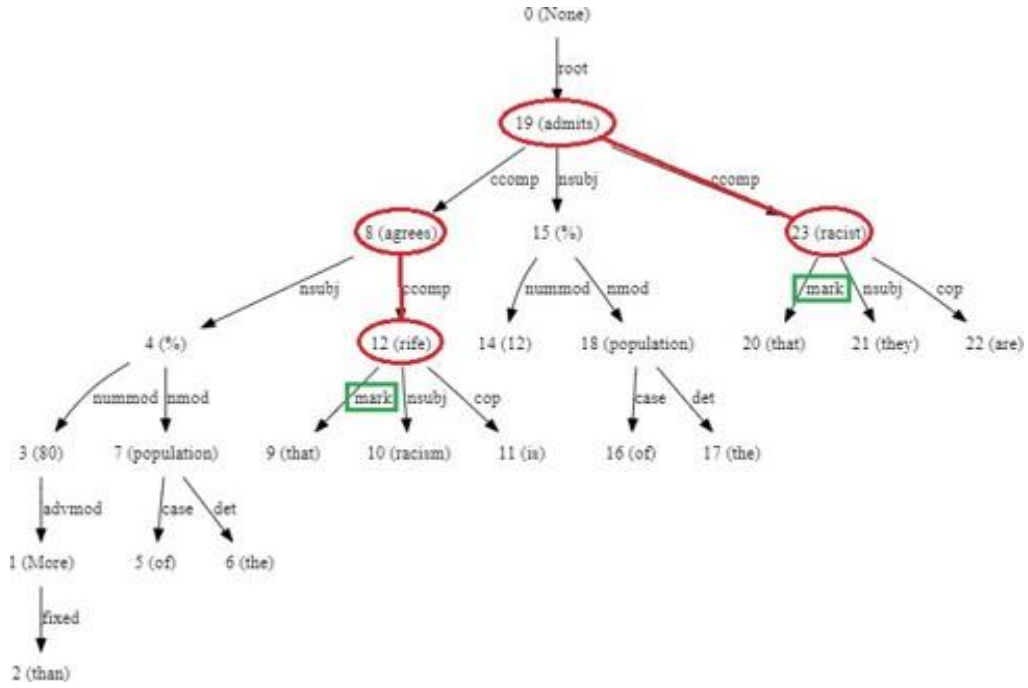
IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Vol. 10, Issue 11, November 2021

DOI: 10.17148/IJARCCE.2021.101131

(a) CSD graph for example 1 (b) CSD graph for example 2



(c) CSD graph7 for example 3

3.1.2 Correction algorithm

If the sentence resulted positive for comma splice, then we need some mechanism to correct it. According to Lamb (1977) comma splice error are resistant to corrections, because it is difficult to predict logical relations between clauses (independent, or dependent) and which conjunction should be used to correct them. So, for now we have simply corrected comma splice by replacing the comma with a period sign. Another, problem in correcting comma splice error is that if we have more than one comma present in the sentence than we need to check which comma is actually comma splice. So, now we describe our correction algorithm which is also based CSD graph.

Step 1: If sentence have only one comma then it is the only comma splice present in the sentence. Otherwise go to STEP 2.

Step 2: Initially mark all commas as **unchecked(UC)**.

Step 3: Iterate over each comma, and perform following sub steps.

(a) Save all such words which are to the left of the current comma in the sentence and which are also CSD nodes, in a list called LEFT.

(b) Similarly, save all such words which are to the right of the current comma in the sentence and which are CSD nodes, in a list called RIGHT.

(c) Now, check in the CSD graph that at least one word(node) from LEFT list is connected with at least one word(node) from RIGHT list. If is it so, then mark comma as **F** (no comma splice) otherwise mark it as **T** (comma splice).

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Vol. 10, Issue 11, November 2021

DOI: 10.17148/IJARCCE.2021.101131

Step 4: Replace **T** marked comma, with a period sign.

Let's see one example, consider this sentence "Amount is very small , and he is rich, he can afford it easily.". See CSD graph of the sentence in figure 5, it is clearly not connected, thus using our detection technique described in section 3.1.1, comma splice is detected, so we applied correction to it using our correction algorithm. Working of each step of correction algorithm is shown in figure 4.

Figure 4: Correction algorithm

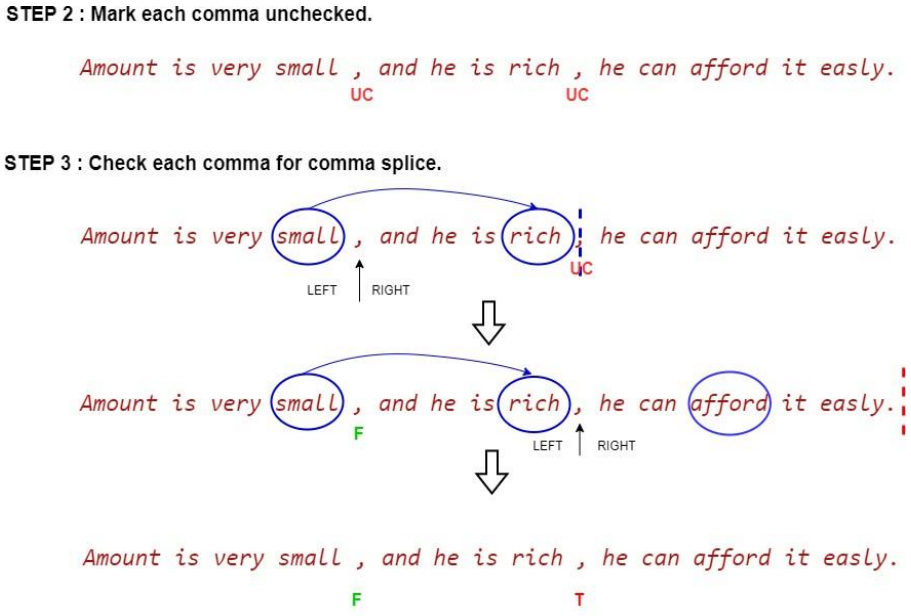
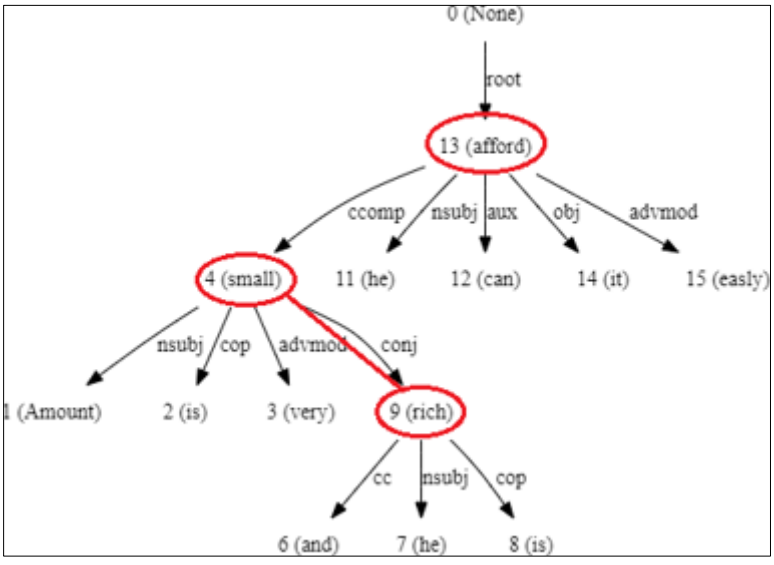


Figure 5: CSD graph



IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Vol. 10, Issue 11, November 2021

DOI: 10.17148/IJARCCE.2021.101131

4. EXPERIMENT AND RESULTS

Testing Set : For testing, we have combined two datasets. One from Ng et al. [7] CoNLL 2014 dataset, which consists 27 errors tagged with error type 'Srun'. Out of which, some are fused sentence error and some are comma splices errors. So, we have selected 21 of them and explicitly converted all of them into comma splice error. Another dataset used for testing is from our previous work Sahu et al. [8]. From this dataset, we have 33 more comma splice errors. Dataset statistics is mentioned in table 1.

Matrices: Since our testing dataset have class imbalance problem, thus to measure performance, we have used these matrices precision (P), recall (R) and F_1 score.

Baseline : To our best knowledge Microsoft Word corrects run on errors (including comma splice errors) so we have compared our system performance with MS Word on-line. We have configured it for run on sentences. To calculate precisions and recalls of MS Word on testing dataset, we have simply pasted sentences into its editor, then we have manually checked each and every suggestions.

Comparison results are shown in Table 2. CSD graph approach out performs. MS word achieved precision 0.07, recall 0.80, and F_1 score 0.12, our CSD graph approach achieved 0.53 precision, 0.28 recall, and 0.36 F_1 score.

Table 1: Testing Dataset

Dataset	**Comma Splice**	**Non Comma Splice**
Ng et al. (2014)	21	1283
Sahu et al. (2020)	33	460
Our Dataset	54	1743

Table 2: Performance comparison

System	**Precision**	**Recall**	F_1 **score**
Microsoft word online	0.07	0.80	0.13
CSD graph approach	**0.53**	**0.28**	**0.36**

5. LIMITATIONS

Some short comings of our approach that we have noticed. 1) As our approach uses dependency parse, if the sentence contains other sentence level errors like fragment error than parser may obtain wrong parse and our approach fails to detect comma splice error. 2) We are unable to correct comma splices in which clauses are joined with a conjunctive adverb ('however', 'thereafter' etc.). This problem can be tackled, via, and adding some more dependencies to connect CSD nodes in graph.

6. CONCLUSION AND FUTURE WORK

In this paper, we have proposed one approach for correcting comma splice errors. Our approach outperforms in comparison to MS word online. In this paper, to fix comma splice errors, we have just replaced comma with a period sign, in future, we will look forward, to also suggest other appropriate corrections (like coordination after the comma or subordination). In future, we will try to apply machine learning to this task.

REFERENCES

[1]. C. Bryant, M. Felice, Ø. E. Andersen, and T. Briscoe. The BEA-2019 shared task on grammatical error correction. In Proceedings of the Fourteenth Workshop on Innovative Use of NLP for Building Educational Applications, pages 52–75, Florence, Italy, Aug. 2019. Association for Computational Linguistics. Doi: 10.18653/v1/W19-4406. https://www.aclweb.org/anthology/W19-4406.

[2]. D. Dahlmeier, H. T. Ng, and S. M. Wu. Building a large annotated corpus of learner English: The NUS corpus of learner English. In Proceedings of the Eighth Workshop on Innovative Use of NLP for Building Educational

International Journal of Advanced Research in Computer and Communication Engineering

Vol. 10, Issue 11, November 2021

DOI: 10.17148/IJARCCE.2021.101131

Applications, pages 22–31, Atlanta, Georgia, June 2013. Association for Computational Linguistics. URL https://www.aclweb.org/anthology/ W13-1703.

[3]. R. Israel, J. Tetreault, and M. Chodorow. Correcting comma errors in learner essays, and restoring commas in newswire text. In Proceedings of the 2012 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, pages 284–294, 2012.

[4]. M. Lamb. An application of error analysis to comma splices and fused sentences. 1977.

[5]. J. Lee, C. Y. Yeung, and M. Chodorow. Automatic detection of comma splices. In Proceedings of the 28th Pacific Asia Conference on Language, Information and Computing, pages 551–560, 2014.

[6]. C. Napoles, M. R. Gormley, and B. Van Durme. Annotated gigaword. In Proceedings of the Joint Workshop on Automatic Knowledge Base Construction and Web-scale Knowledge Extraction (AKBC-WEKEX), pages 95–100, 2012.

[7]. H. T. Ng, S. M. Wu, T. Briscoe, C. Hadiwinoto, R. H. Susanto, and C. Bryant. The conll-2014 shared task on grammatical error correction. In Proceedings of the Eighteenth Conference on Computational Natural Language Learning: Shared Task, pages 1–14, 2014.

[8]. S. Sahu, Y. K. Vishwakarma, J. Kori, and J. S. Thakur. Evaluating performance of different grammar checking tools. International Journal, 9(2), 2020.

[9]. Soni, Madhvi, and Jitendra Singh Thakur. "A systematic review of automated grammar checking in English language." *arXiv preprint arXiv:1804.00540* (2018).

[10]. I. Sutskever, O. Vinyals, and Q. V. Le. Sequence to sequence learning with neural networks. In Advances in neural information processing systems, pages 3104–3112, 2014.

[11]. O. Tilk and T. Alumäe. Bidirectional recurrent neural network with attention mechanism for punctuation restoration. In Interspeech, pages 3047–3051, 2016.

[12]. J. Zheng, C. Napoles, J. Tetreault, and K. Omelianchuk. How do you correct run-on sentences it's not as easy as it seems. arXiv preprint arXiv:1809.08298, 2018

