Efficient Dependency Tree Sampling Without Replacement

Bogdan Mihai Dobre
University of Bucharest
bogdan.dobre@s.unibuc.ro

Abstract

In the context of computational models of dependency syntax, most dependency treebanks have the restriction that any valid dependency tree must have exactly one edge coming out of the root node in addition to respecting the spanning tree constraints. Many algorithms for dependency tree sampling were recently proposed, both for sampling with and without replacement.

In this paper we propose a new algorithm called Wilson Reject SWOR for the case of sampling without replacement by adapting the Wilson Reject algorithm originally created for sampling with replacement and combining it with a Trie data structure. Experimental results indicate the efficiency of our approach in the scenario of sampling without replacement from dependency graphs with random weights.

1 Introduction

Dependency trees are one of the most popular structures used to represent syntactic relations between words (Kübler et al., 2009). A popular formalisation for them is based on spanning trees in directed graphs, which is one of the core ways to represent dependency structures in natural language processing (McDonald et al., 2005). A dependency graph for n words has n+1 nodes: one for each word and a special root node. It also has n^2 weighted edges between nodes: n(n-1) edges between any pair of distinct word nodes and n edges from the root to each node. A spanning tree of the graph is a subgraph that contains all the n+1 nodes, exactly n edges and no cycles. Each node except for the root has exactly one incoming edge and is reachable from the root. Probabilistic models over spanning trees have uses in non-projective dependency parsing (Ma and Hovy, 2017; Dozat and Manning, 2016; Wang and Chang, 2016), recovering phylogenic structures (Andrews et al., 2012) and event extraction (McClosky et al., 2011).

A typical dependency tree constraint that is also applied in the case of Universal Dependencies (Nivre et al., 2020) is that the tree must have exactly one edge coming out of the root node. In the rest of the paper we will refer to spanning trees that have exactly one edge coming out of the root node as "dependency trees".

The need for algorithms that sample dependency trees comes from the fact that state-of-the-art parsers often predict invalid dependency trees as the most probable output (Zmigrod et al., 2020).

Dependency tree sampling has uses in domains such as semi-supervised training of parsers (Corro and Titov, 2018), facilitating the approximate decoding of higher-order models (Zhang et al., 2014) and unsupervised grammar induction (Mareček and Žabokrtský, 2011).

There are 2 main sampling scenarios: sampling with replacement and sampling without replacement. In the case of sampling with replacement, the same tree can be sampled multiple times, while in the case of sampling without replacement, each tree can only be sampled once. Some recent advances in dependency trees sampling can be found in (Stanojević, 2022), including algorithms for sampling dependency trees both with and without replacement, but in this paper we will focus on adapting the fastest algorithm in practice for sampling with replacement, Wilson Reject, to sampling without replacement, thus creating a new algorithm that we will call Wilson Reject SWOR that achieves superior performance compared to the other algorithms for graphs with random weights. The random weights setting usually occurs in the beginning of training of dependency parsers.

2 Distributions over tress

In this section we will give formal definitions to distributions over spanning trees and dependency trees in rooted directed weighted graphs. A rooted directed graph can be denoted by G=$\{V, E\}$, where

V is a set of n+1 nodes (1 node for each of the n words and one special root node) and E is a set of directed weighted edges of the form $\{n_1, n_2, w\}$ where n_1 is the starting node of the edge, n_2 is the end node of the edge and w is the nonnegative weight of an edge.

A spanning tree is any subgraph of G that contains all the n+1 nodes, has exactly one incoming edge for each node except for the root and contains no cycles.

A dependency tree is any subgraph of G that is a spanning tree and also has exactly one edge coming out of the root node. Let D_G be the set of all the dependency trees of G.

Let's denote the edge from node i to node j with $i \to j$ and the weight of the edge from node i to node j with $W_{(i,j)}$.

Weight of a dependency tree can then be defined as:

$$W_t \overset{\text{def}}{=} \prod W_{(i,j)}, i \to j \in t$$

Probability of a dependency tree can similarly be defined as:

$$p(t) \overset{\text{def}}{=} \frac{W_t}{Z_{D_G}}, \text{ where } Z_{D_G} \overset{\text{def}}{=} \sum_{t \in D_G} w(t)$$

Any unbiased dependency tree sampler must sample the trees according to their probabilities.

3 Previous Work on Sampling Dependency Trees

For the problem of sampling with replacement (SWIR), the fastest algorithm in practice is the Wilson Reject algorithm (shown in Algorithm 2), proposed here (Stanojević, 2022). Wilson Reject is based on rejection sampling, a strategy that we will also use in the algorithm we propose. Rejection sampling is commonly used when we want to sample from a distribution $d_1(t)$ from which sampling is difficult, but there is a related distribution $d_2(t)$ from which sampling is easier. More details about rejection sampling can be found in (Murphy, 2012, Chapter 23.3). Wilson Reject uses Wilson's sampling algorithm to sample spanning trees until a valid dependency tree is sampled. Wilson's algorithm is a random walk based algorithm. The algorithm starts with the root node as the current selection. At each step, if there is a node that is not in the current selection, it starts a random walk from that node through its incoming edges until it reaches a node in the current selection. During the walk it samples a parent node of the current node randomly based on the weights of the incoming edges of the node. Any cycles during the walk are implicitly deleted by the algorithm. The resulting chain is attached to the current selection. When all nodes are part of the selection, the selection will be a spanning tree and will be returned by the algorithm. It was proved in (Stanojević, 2022) that Wilson Reject is an unbiased sampler, and that the expected number of samples it needs until it gets a valid dependency tree is Euler's number. Therefore, 3 runs of Wilson's sampling algorithm are expected to be enough to sample a valid dependency tree. The total runtime complexity of Wilson's algorithm depends on the graph edges and their weights, but can be as small as $O(|V|)$ (Wilson, 1996). Therefore, the complexity of Wilson Reject is also often as small as $O(|V|)$.

The pseudocode of Wilson's Algorithm can be found in Algorithm 1, and the pseudocode of Wilson Reject can be found in Algorithm 2.

Algorithm 1 WILSON

```
1: $p \leftarrow \emptyset$    ▷ $p$ will keep the sampled tree edges
2:                            ▷ $p[n_1] = n_2$ stands for edge $n_2 \to n_1$
3: $visited \leftarrow \{ROOT\}$
4: for $i \in V \setminus \{ROOT\}$ do
5:     $c \leftarrow i$
6:     while $c \notin visited$ do
7:         random sample node $v$ with edge weight $W_{v \to c}$
8:         $p[c] \leftarrow v$
9:         $c \leftarrow v$
10:    $c \leftarrow i$
11:    while $c \notin visited$ do
12:        $visited$.add($c$)
13:        $c \leftarrow p[c]$
14: return $p$
```

Algorithm 2 WILSON REJECT

```
1: $t \leftarrow$ Wilson($G$)
2: while $t$ has more than one root edge do
3:     $t \leftarrow$Wilson($G$)
4: return $t$
```

4 Wilson Reject SWOR

We will now analyze the scenario of sampling without replacement (SWOR), scenario in which it is not allowed to sample the same instance twice. For this scenario, 2 algorithms with complexity $O(k * n^3)$ where k is the number of samples and n the number of words were proposed in (Stanojević, 2022): Trie-SWOR and SBS-SWOR. Both use Colbourn's algorithm (Colbourn et al., 1996), an ancestral sampling algorithm, as an auto-regressive form of dependency tree distribution and each of the 2 algorithms uses a different algorithm for sampling from sequential models: Trie-SWOR is based on the Trie algorithm (Shi et al., 2020) and SBS-SWOR is based on Stochastic Beam Search (SBS) (Kool et al., 2019). More details about these algorithms can be found in (Stanojević, 2022).

We will focus on comparing our results with the SBS-SWOR algorithm, as it was found to be faster than Trie-SWOR on CPU and, unlike Trie-SWOR, its performance improves significantly when run on GPU. This is mainly caused by the fact that SBS-SWOR draws all the required samples in parallel instead of sequentially, so it can substantially benefit from the GPU's parallel computing. In order to ensure that no instance is sampled twice, SWOR algorithms need to constrain the current sample based on the previous samples (Kool et al., 2020).

We create a new algorithm called Wilson Reject SWOR based on the Wilson Reject algorithm for the random weights setting. We do that by keeping a list of already sampled trees that are considered invalid samples for this step. At each sampling step we keep generating dependency trees with the Wilson Reject algorithm until we get a tree that wasn't already sampled. That tree is considered the valid sample for that step and added to the list of already sampled trees.

Theorem 1. *Wilson Reject SWOR is unbiased.*

Proof. We have to prove that at each step Wilson Reject SWOR samples dependency trees in an unbiased way. We consider that the trees sampled before the current step are invalid as they can't be sampled again in SWOR. It was already proved that Wilson Reject is an unbiased sampler of dependency trees (Stanojević, 2022). We can use rejection sampling concepts (Murphy, 2012, Chapter 23.3) to prove that Wilson Reject SWOR which uses Wilson Reject is unbiased. Let C be the current step, S_C be the set of already sampled dependency trees, $d_{1_{S_C}}(t)$ be the distribution of valid dependency trees at step C and $d_2(t)$ be the distribution of all dependency trees that can be sampled. The condition for rejection sampling to work is that there must be a constant $c > 0$ such that $c * d_2(t) \geq \tilde{d}_{1_{S_C}}(t)$, where $\tilde{d}_{1_{S_C}}(t)$ is the unnormalized target distribution. After a sample is retrieved from $d_2(t)$, it is accepted with probability $\frac{\tilde{d}_{1_{S_C}}(t)}{c * d_2(t)}$. In our case, since the trees in both distributions come from the same graph, they have the same weights, so we can have $c = 1$ and accept any sample from $d_2(t)$ that is also part of $d_{1_{S_C}}(t)$. Therefore, accepting any dependency tree sampled by Wilson Reject that wasn't already sampled is an unbiased SWOR sampling method. This means that Wilson Reject SWOR is unbiased. □

Any node in a dependency tree except for the root can have exactly one incoming edge. Therefore, if we have an edge from a node n_1 to a node n_2, we will call n_1 the "parent" of node n_2. To make the Wilson Reject SWOR algorithm more efficient, we can uniquely represent any dependency tree using a list of the parents of all the nodes. Since the root has no parent in a dependency tree, we use a placeholder for its parent in the list. Building a list of parents from a dependency tree and building a dependency tree from a list of parents are trivial operations that have $O(|V|)$ time complexity.

We can keep the parents list representations of the already sampled trees in a Trie data structure. This allows us to check if a tree was already sampled by searching for its associated parents list in the Trie in $O(|V|)$ time. Insertion of the parents list associated to a tree in the Trie can also be done in $O(|V|)$ time (Brass, 2008). Wilson Reject SWOR has no worst case complexity, but we can estimate the complexity for graphs with random weights. An important question is how often do we expect to sample a tree that was already sampled, which depends on the number of nodes, the configuration of edges and the number of samples. It is well known that there are n^{n-2} spanning trees for a complete graph of size n (Cayley, 1878), and from each of them we can build a dependency tree for the graph with n+1 nodes corresponding to n words by adding the artificial root node and an edge from it to the root of the spanning tree. Therefore, for the graph of size n+1 corresponding to n words, there are n^{n-2} dependency trees. Since each dependency tree has the same number of edges, for graphs with random edge weights we can expect most dependency trees to most likely have their total weights close to their mean. This indicates that for graphs with a significant number of nodes and random weights, it is very unlikely the same tree would be sampled twice without a very big number of samples. As an experiment, we sampled 20000 dependency trees from a graph of size 100, and the same tree was never sampled twice. For cases when the same tree is only sampled once, we can consider that the complexity of Wilson Reject SWOR is the same as that of Wilson Reject, which as discussed previously depends on many factors but can be as small as $O(|V|)$. Therefore, the complexity of Wilson Reject SWOR can be as small as $O(k * |V|)$, where k is the number of samples. The steps of the algorithm are shown in Algorithm 3:

Algorithm 3 Wilson Reject SWOR

```
1: $G \leftarrow$ original graph
2: $t \leftarrow$ sample_with_wilson_reject($G$)
3: $t\_parent\_array \leftarrow$ get_prnt_array($t$)
4: $trie \leftarrow \emptyset$
5: **while** $t\_parent\_array \in trie$ **do**:
6:     $t \leftarrow$ sample_with_wilson_reject($G$)
7:     $t\_parent\_array \leftarrow$ get_prnt_array($t$)
8: $trie$.insert($t\_parent\_array$)
9: **return** $t$
```

5 Experiments

We will now compare the results of Wilson Reject SWOR to those of SBS-SWOR for the case of sampling without replacement. Unlike Wilson Reject SWOR, SBS-SWOR can be significantly improved by GPU acceleration due to the fact that it draws all samples in parallel, so it makes sense to compare the results with both those of SBS-SWOR on CPU and those of SBS-SWOR on GPU. The experiments were run on a 11th Gen Intel® Core(TM) i7-11800H @ 2.30GHz CPU and a NVIDIA® GeForce RTX 3050 GPU. The SBS-SWOR implementation that was used was from the SynJax Python library, version 2023.8.5, (Stanojević and Sartran, 2023) which is written in JAX (Bradbury et al., 2018). We will present the results of the 2 algorithms for dependency graphs with random weights corresponding to 100 words. We found that Wilson Reject SWOR on CPU has better performance than both SBS-SWOR on CPU and SBS-SWOR on GPU, as it can be seen in Figure 1. It also has the advantage of being sequential unlike SBS-SWOR, allowing us to stop sampling if certain criteria were met, and the advantage of not relying on GPU. Results for additional experiments with graphs with different numbers of words can be found in Appendix A.

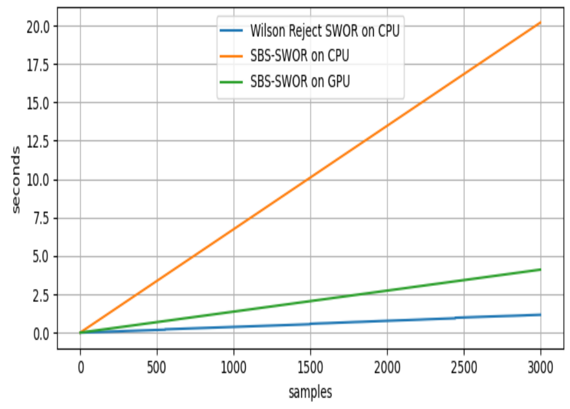

Figure 1: Speed for SWOR with 100 words

6 Conclusion

In this paper we presented the Wilson Reject SWOR algorithm for sampling dependency trees without replacement which is based on random walks. We also showed that it is unbiased and has the best performance for graphs with random weights. It may have uses in tasks where dependency tree sampling is required such as semi-supervised training of parsers (Corro and Titov, 2018), unsupervised grammar induction (Mareček and Žabokrtský, 2011) and approximate decoding of higher-order models (Zhang et al., 2014).

Limitations

Even though the Wilson Reject SWOR algorithm is very efficient for sampling dependency trees without replacement for graphs with random weights, it is still based on random walks due to its reliance on the original Wilson's Algorithm. That means that we don't have any worst case complexity for it. Methods such as Colbourn based algorithms have a predictable runtime, even though in practice they tend to be slower. There are some types of graphs for which the performance is expected to be greatly impacted, such as graphs in which almost all of the probability mass is on a single dependency tree. For this kind of graphs we can expect that a big number of samples would be required to sample anything but the most probable dependency tree, making Wilson Reject SWOR inefficient for these cases compared to alternatives like SBS-SWOR. Therefore, the algorithm would be most useful in the random weights setting that usually happens in the beginning of training of dependency parsers.

Ethics Statement

To the best of our knowledge, there isn't any way in which the work presented in this paper could raise ethical concerns.

References

Nicholas Andrews, Jason Eisner, and Mark Dredze. 2012. Name phylogeny: A generative model of string variation. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 344–355.

James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. 2018. JAX: composable transformations of Python+NumPy programs.

Peter Brass. 2008. *Advanced Data Structures*. Cambridge University Press.

Arthur Cayley. 1878. A theorem on trees. *Quart. J. Math.*, 23:376–378.

Charles J Colbourn, Wendy J Myrvold, and Eugene Neufeld. 1996. Two algorithms for unranking arborescences. *Journal of Algorithms*, 20(2):268–281.

Caio Corro and Ivan Titov. 2018. Differentiable perturb-and-parse: Semi-supervised parsing with a structured variational autoencoder. *arXiv preprint arXiv:1807.09875*.

Timothy Dozat and Christopher D Manning. 2016. Deep biaffine attention for neural dependency parsing. *arXiv preprint arXiv:1611.01734*.

Wouter Kool, Herke Van Hoof, and Max Welling. 2019. Stochastic beams and where to find them: The gumbel-top-k trick for sampling sequences without replacement. In *International Conference on Machine Learning*, pages 3499–3508. PMLR.

Wouter Kool, Herke van Hoof, and Max Welling. 2020. Estimating gradients for discrete random variables by sampling without replacement. *arXiv preprint arXiv:2002.06043*.

Sandra Kübler, Ryan McDonald, and Joakim Nivre. 2009. Dependency parsing. In *Dependency parsing*, pages 11–20. Springer.

Xuezhe Ma and Eduard Hovy. 2017. Neural probabilistic model for non-projective mst parsing. *arXiv preprint arXiv:1701.00874*.

David Mareček and Zdeněk Žabokrtský. 2011. Gibbs sampling with treeness constraint in unsupervised dependency parsing. In *Proceedings of Workshop on Robust Unsupervised and Semisupervised Methods in Natural Language Processing*, pages 1–8, Hissar, Bulgaria. Association for Computational Linguistics.

David McClosky, Mihai Surdeanu, and Christopher D Manning. 2011. Event extraction as dependency parsing for bionlp 2011. In *Proceedings of BioNLP Shared Task 2011 Workshop*, pages 41–45.

Ryan McDonald, Fernando Pereira, Kiril Ribarov, and Jan Hajič. 2005. Non-projective dependency parsing using spanning tree algorithms. In *Proceedings of Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing*, pages 523–530, Vancouver, British Columbia, Canada. Association for Computational Linguistics.

Kevin P Murphy. 2012. *Machine learning: a probabilistic perspective*. MIT press.

Joakim Nivre, Marie-Catherine de Marneffe, Filip Ginter, Jan Hajič, Christopher D. Manning, Sampo Pyysalo, Sebastian Schuster, Francis Tyers, and Daniel Zeman. 2020. Universal Dependencies v2: An evergrowing multilingual treebank collection. In *Proceedings of the Twelfth Language Resources and Evaluation Conference*, pages 4034–4043, Marseille, France. European Language Resources Association.

Kensen Shi, David Bieber, and Charles Sutton. 2020. Incremental sampling without replacement for sequence models. In *International Conference on Machine Learning*, pages 8785–8795. PMLR.

Miloš Stanojević. 2022. Unbiased and efficient sampling of dependency trees. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 1691–1706.

Miloš Stanojević and Laurent Sartran. 2023. SynJax: Structured Probability Distributions for JAX. *arXiv preprint arXiv:2308.03291*.

Wenhui Wang and Baobao Chang. 2016. Graph-based dependency parsing with bidirectional lstm. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2306–2315.

David Bruce Wilson. 1996. Generating random spanning trees more quickly than the cover time. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, pages 296–303.

Yuan Zhang, Tao Lei, Regina Barzilay, and Tommi Jaakkola. 2014. Greed is good if randomized: New inference for dependency parsing. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1013–1024.

Ran Zmigrod, Tim Vieira, and Ryan Cotterell. 2020. Please mind the root: Decoding arborescences for dependency parsing. *arXiv preprint arXiv:2010.02550*.

A Results for additional SWOR experiments

In this appendix we will present additional results for Wilson Reject SWOR and SBS-SWOR on graphs with different word counts. The experiments were run on a 11th Gen Intel® Core(TM) i7-11800H @ 2.30GHz CPU and a NVIDIA® GeForce RTX 3050 GPU and the SBS-SWOR implementation that was used was from the SynJax Python library, version 2023.8.5, (Stanojević and Sartran, 2023) which is written in JAX (Bradbury et al., 2018). We found that Wilson Reject SWOR on CPU achieved a better performance than SBS-SWOR on CPU and SBS-SWOR on GPU for all the word counts tried for graphs with random weights, thus proving its efficiency.

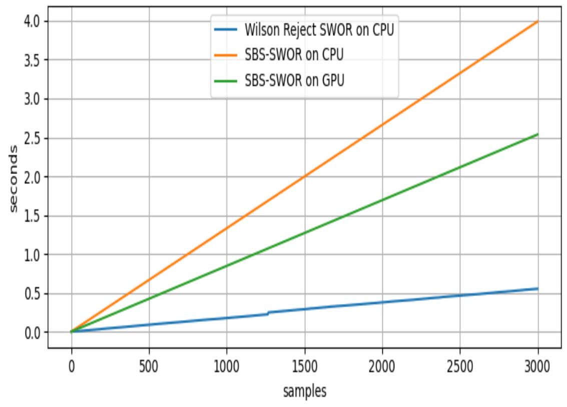

Figure 2: Speed for SWOR with 45 words

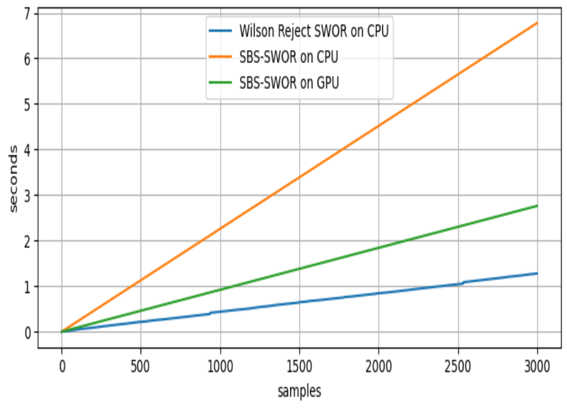

Figure 3: Speed for SWOR with 60 words

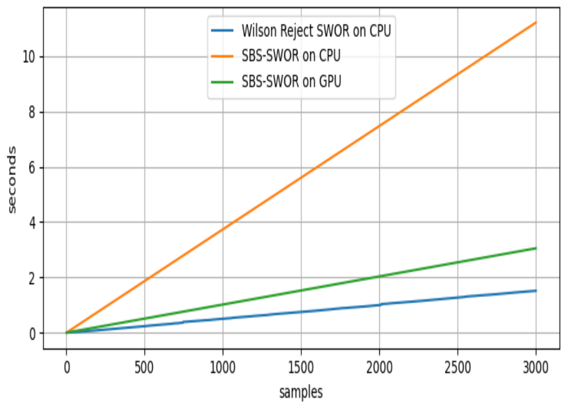

Figure 4: Speed for SWOR with 75 words

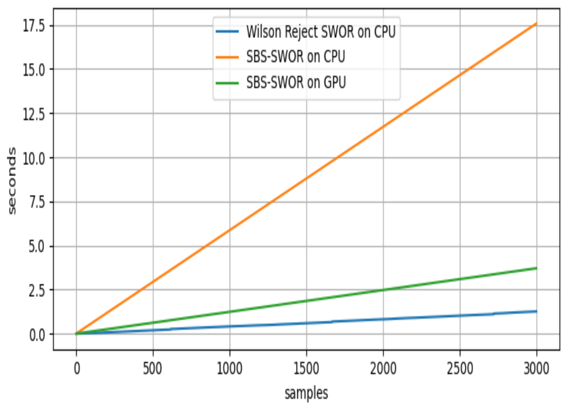

Figure 5: Speed for SWOR with 90 words

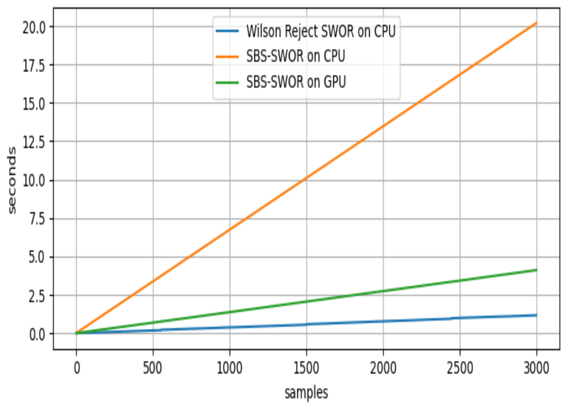

Figure 6: Speed for SWOR with 100 words