Proceedings of the 13th Conference on Language Resources and Evaluation (LREC 2022), pages 5054–5062
Marseille, 20-25 June 2022

A Learning-Based Dependency to Constituency Conversion Algorithm for the Turkish Language

Büşra Marşan[♡], Oğuz Kerem Yıldız[♠], Aslı Kuzgun[♡], Neslihan Cesur[♡], Arife Betül Yenice[♡]
Ezgi Sanıyar[♡], Oğuzhan Kuyrukçu[♡], Bilge Nas Arıcan[♡], Olcay Taner Yıldız[♢]
Starlang Yazılım Danışmanlık[♡], Ahmet Keleşoğlu High School[♠], Ozyegin University[♢]
Istanbul, Turkey
{busra, oguz, asli, neslihan, arife, ezgi, oguzhan, bilge}@starlangyazilim.com, olcay.yildiz@ozyegin.edu.tr

Abstract

This study aims to create the very first dependency-to-constituency conversion algorithm optimised for Turkish language. For this purpose, a state-of-the-art morphologic analyser (Yıldız et al., 2019) and a feature-based machine learning model was used. In order to enhance the performance of the conversion algorithm, bootstrap aggregating meta-algorithm was integrated. While creating the conversation algorithm, typological properties of Turkish were carefully considered. A comprehensive and manually annotated UD-style dependency treebank was the input, and constituency trees were the output of the conversion algorithm. A team of linguists manually annotated a set of constituency trees. These manually annotated trees were used as the gold standard to assess the performance of the algorithm. The conversion process yielded more than 8000 constituency trees whose UD-style dependency trees are also available on GitHub. In addition to its contribution to Turkish treebank resources, this study also offers a viable and easy-to-implement conversion algorithm that can be used to generate new constituency treebanks and training data for NLP resources like constituency parsers.
Keywords: Constituency parsing, Dependency parsing, Constitueny Dpendency conversion

1. Introduction

Following the phrase-based translation model (Koehn et al., 2003), syntactic trees have been prominent in the past studies on statistical grammar (Ding and Palmer, 2005; Ding and Palmer, 2004), statistical parsing (Grammar and Hockenmaier, 2003; Wang and Harper, 2004; Cahill et al., 2008; Candito et al., 2010), machine translation (Huang et al., 2006; Ding and Palmer, 2005; Shen et al., 2008; Xie et al., 2011) and various other subbranches of NLP. There are two main syntactic tree representations: Dependency and constituency trees. Both are able to capture valency, argument structure and various other syntactic relations between phrases. Although it is possible to argue that dependency trees are better equipped to handle long-distance dependencies and free word order languages, constituency trees can better illustrate phrasal continuity and government.

Based on the notion of dependency grammar first introduced by Lucien Tesnière (See (Percival, 1990) for a detailed history and discussion), dependency trees connect linguistic units (heads and dependents) through directed links and labels these links based on the relationship between the head and its dependent(s). Inspired by term logic, constituency trees have their theoretical origins at phrase structure grammars (also known as constituency grammars) presented by Noam Chomsky (Chomsky, 1957). Based on constituency relation, constituency trees illustrate phrases as sub-trees and shows the relationship between not only the head and its dependent but also the relationship between the dependents of the same head (See Figure 2). Although these two syntactic representations appear very distinct

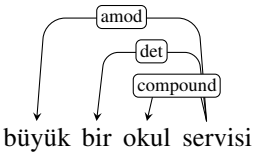

Figure 1: Example dependency tree
("a big school bus")

at first sight (refer to (Matthews, 1981) for a thorough discussion), they share enough features that they can be combined (Sag et al., 2003), or even converted into one another (Kong et al., 2015). With the Universal Dependencies' (Nivre et al., 2020) attempt to offer a framework and set of rules that can be applied to a wide range of different languages, an impressively comprehensive multilingual dependency treebank[1] was created to offer much needed data -especially for low resource languages.

Syntactic treebanks, both dependency and constituency, are essential resources for many areas of study including linguistic research, natural language processing, machine translation, machine learning and improving pre-trained transformers (Bai et al., 2021) yet their possible applications and practical value are "potentially limited by the degree to which they subscribe to a specific linguistic theory" (Bick, 2006). That is why being able to convert such treebanks into another format is so important that it must also be considered during the design processes of these treebanks (Nivre, 2003). Moreover, conversions between

[1] https://github.com/UniversalDependencies

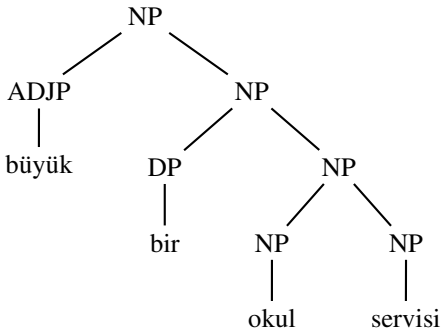

Figure 2: Example constituency tree

constituency and dependency trees allow researchers to accommodate advantages and disadvantages of each structure, such as dependency's ability to better illustrate clefting, non-projectivity and raising, or constituency's firmer grasp on coordination (Bick, 2006).

In addition, treebanks are often employed in creating and evaluating automatic syntactic parsers. As stated by Daum et al. (2004), practice of training and evaluating such parsers gives rise to the need for massive quantities of annotated trees (Daum et al., 2004). Also, extensive treebanks are very useful in building rule governed natural language processing tools and systems (Bouma and Kloosterman, 2002), context free grammars and statistical grammars. That is why a successful dependency to constituency conversion algorithm is especially valuable for low resource languages like Turkish.

For the reasons discussed above, many dependency to constituency and constituency to dependency conversion algorithms have been developed in the last two decades (Höfler, 2002; Lee and Wang, 2016; Bosco, 2007; Bick, 2006; Xia and Palmer, 2001; Lu et al., 2016; Kong et al., 2015). The majority of such algorithms work with CoNLL-U, Stanford Dependencies and UD. Constituency trees created by such algorithms often follow Penn Treebank format or a specific linguistic framework like X-Bar Theory (Chomsky, 1968). One of the trends in dependency to constituency conversion studies is using algorithm generated dependencies. The downside of this method stems from the very nature of dependency trees. Dependency structures are defined by head dependent relations yet in some rather language specific cases, the decision on the head of a phrase can be contestable. The phrase "genel olarak" *(lit. "general, to be"; "generally, broadly")* is a perfect example for that (Figure 3). "olarak" ('$\backslash being$") is a light verb in Turkish, hence some linguists consider it an auxiliary and argue that the head of "genel olarak" is "genel" as Dependency Framework doesn't allow auxiliaries or function words to be phrasal heads. Following this insight, the dependency tree of "genel olarak" looks like (A) below. On the other hand, it i possible to argue that "olarak" is the verbal head and "genel" is an adverb (B). Due to its high frequency, some linguists argue that speakers do not decompose this expression, rather treat it as a compound. Hence, another possibility is to draw a dependency tree like (C) for "genel olarak." When annotated manually, annotators can discuss and agree upon how to annotate such cases to ensure the consistency in theoretical framework and inter-annotator agreement. Yet algorithm generated dependency treebanks often lack coherency in such language specific cases where head - dependent relation is disputable.

One of the biggest aims of this study was to create a constituency treebank for Turkish, a low resource language. For this purpose, we used an existing Universal Dependency treebank [2]. The reasons behind our choice were:

- Our desire to use a manually annotated Dependency treebank for the reasons we discussed above.
- Offering an enhanced dataset that has both Dependency and constituency trees.

A learning-based algorithm that takes in Universal Dependency trees and converts them into constituency trees is created for this study. This conversion algorithm takes the dependency structure as its input to iterate all links and find all phrasal projections in the structure. After detecting the phrases, the task is merging the sub-trees of these phrases in the right hierarchical order to create the constituency tree. In order to do so, it the algorithm does one of the three operations: Left, Right and Merge. Left operation fuses the node at hand with the one on the immediate left. Right operation does the same with the node on the immediate right. Merge operation merges all nodes at hand to create a phrasal projection. The goal of the algorithm is deciding upon the correct operations and applying them in the correct order to create the desired constituency tree. As the very first study of dependency to constituency conversion in Turkish, the work presented in this paper sets a baseline and offers a swift and accurate method to enrich constituency tree datasets in Turkish using Universal Dependency trees.

This paper is organised as follows: First the conversion process is explained in terms of input and machine learning-based conversion algorithm. Following the conversion process, the evaluation of the conversion algorithm's performance is discussed and finally, closing remarks are made.

2. Basic Structure of the Constituency Trees

In this study, we opted for an algorithm that produces flatter phrase structures compared to the X-Bar the-

[2]https://github.com/UniversalDependencies/UD_Turkish-Penn

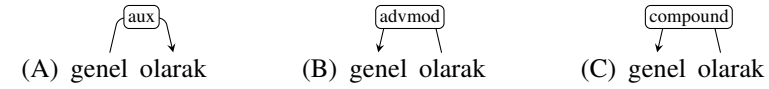

Figure 3: Possible dependency annotations of "genel olarak" *(generally, broadly)*

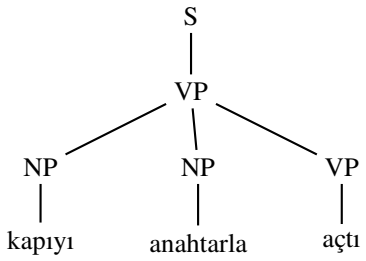

Figure 4: "[S/he] opened the door with a key"

ory representations (Chomsky, 1968) and Penn TreeBank trees by allowing ternary branching and bypassing bar-levels. Flatter trees are useful for reflecting the argument structure of the predicate since they are able to show all complements in the same level. This way of representation allows more specific context-free grammar rules, thus it can be used in CFG applications as well. There is a Turkish PropBank as a resource for a better analysis of the predicate features (Kara et al., 2020). Incorporating a PropBank can enhance the accuracy of phrase structures and trees that a conversion algorithm offers. This way, constituency trees can be more linguistically sound since they directly reflect the adjunct and argument structure of the predicate and, as a bonus, this methodology is useful for conversions to other languages as well, since it does not depend on the language specific restrictions or rules and operates on the predicate itself.

To make trees flat, we followed the strategy also employed by Hajic et al. (1998):

- Each constituent X has only one parent, XP.
- There is no X' level or its equivalent (such as XX).

Once the head of a phrase is detected, all of its dependents are attached on the same level. In other words, the algorithm does not assign different positions for a complement and an adjunct. This feature is illustrated in the tree below (Figure 4) where "kapıyı" *("the door")* is a complement and "anahtarla" *("with a key")* is an adjunct of the verb "açtı" *("opened")*.

Our trees diverge from the Penn TreeBank's trees mainly due to the tags used for the subtrees and empty projections. Contrary to the Penn format, the baseline algorithm produces trees that have no bar-level projections and empty projections. We decided that the bar level production is not necessary for properly reflecting the argument structure of the verb since there is also a Turkish PropBank ready (Kara et al., 2020). This decision led us to flatter trees. In addition, we based the phrasal tags on morphology instead of the syntactic and/or semantic roles, meaning that phrasal projections have basic tags like NP, PP and VP based on the morphological features. Our constituency trees lack semantic function indicators like (NP)-SUBJ, (NP)-OBJ, or (ADV)-TMP. We used morphological analyser to identify the morphological features and POS tags for the morphemes (Yıldız et al., 2019).

3. Conversion Process

3.1. Input

The input of the conversion algorithm is a manually annotated dependency treebank which can be found in a GitHub repository of Universal Dependencies organisation[3]. The data of this treebank consists of Penn Treebank sentences, translated into Turkish by professional translators. After being proofread and edited, these translated sentences were manually annotated by a team of linguists following the UD framework and style guide. In order to ensure inter-annotator agreement and quality, manually annotated dependencies were revised and edited when necessary by two linguists after the annotation process.

3.2. Conversion Algorithm

The conversion algorithm follows a bottom-up approach to start conjoining terminal nodes and adding their phrasal projections until it reaches the root. Along the process, dependency tags and headedness indicated in the dependency trees are referred to correctly convert each dependency relation. For each edge in the dependency tree, a sub-tree is generated. Then these subbranches are conjoined using the dependency relations, headedness, and POS tags indicated in the dependency trees. POS tags determine the phrasal tags such as NP, VP, DP and so forth.

There are two different oracles created for the purposes of this study, one of which refers to a set of predefined heuristics to conjoin subtrees. These predefined set of heuristics are based on the dependency relations, POS tags and headedness. These rules guide sub-tree merger process. After every terminal node is assigned to a sub-tree in accordance with it headed or dependent, the basic oracle refers to the rules and merges them by one of the three operations: Left, Right and Merge. The operations applied at this stage and their

[3]https://github.com/UniversalDependencies/UD Turkish-Penn

order are critical to create the correct tree output. Hence, the algorithm refers to two oracles to make the correct decision:

- The first oracle is the basic oracle which bases merging decisions on the set of rules discussed above, which are carefully created by a team of linguists in accordance with UD framework and syntactic typology of Turkish language. Trees created by the basic oracle constitutes the baseline for this study. The learning-based algorithm aims to exceed the performance of this basic oracle.
- The second one is the classifier oracle which bases merging decisions on a machine learning model.

After the subtree merging operations are complete, these subtrees are linked to S node, thus creating the final constituency tree in the final step. For the learning-based oracle, the number of dependents the head has guides the subtree merging process. Groups for 1-dependent, 2-dependent, 3-dependent and 4-dependent instances are first determined. Then combinatorial constraints that refer to dependency relations and syntactic typology of Turkish are created for each group. In accordance with these constraints, the learning-based algorithm performs three operations (Left, Right and Merge) to combine sub-trees and create the final constituency tree output.

3.2.1. Basic Oracle

Basic oracle is exclusively rule-based. In other words, it makes merging decisions in accordance with a set of rules and heuristics created by a team of linguists. Basic oracle was created and used solely for setting the baseline for this study. The machine learning based conversion algorithm only uses the classifier oracle and does not refer to the basic oracle at any point of the conversion process.

The main challenge for the basic oracle is grouping the subtrees in the correct order. For example the verb "merak etmek" (lit. "curious, to do" *"to wonder something")* is a transitive light verb construction. To appropriately reflect its argument structure, the oracle needs to group the words "merak" and "etmek" as a VP (this VP is the predicate) and then it needs to append the object NP as a sister to the VP. After merging the object, the algorithm needs to append adjuncts and adverbs to VP, if there are any. This order of merger operation is essential since if the adjuncts are merged before the object(s), the algorithm yields an incorrect tree. Thus, a hierarchy of subtree appendage is created to avoid doing the merger operations in the wrong order. If the dependency tag of a subtree is COMPOUND, it is appended to the S first. Then comes AUX, DET, AMOD, NUMMOD, CASE, CCOMP and finally NEG (See Figure 5). The reason why NEG ("değil," a negation particle exclusively used with nominal predicates) is at the bottom of the hierarchy is the fact that it has to

COMPOUND >AUX >DET >AMOD > NUMMOD >CASE >CCOMP >NEG

Figure 5: The hierarchy of subtree appendage

govern and C-Command the entire predicate it negates. COMPOUND is the highest in the hierarchy to maintain the integrity of phrasal verbs, light verb constructions, and other two-word expressions. The rest of the hierarchy is worked out by a team of linguists in accordance with the typological features of Turkish.

Following the rule-based basic oracle, merging takes place as follows:

In order to create the subtrees, the algorithm first refers to the morphological analyser which provides basic tags like N, ADJ and ADV. These basic tags constitute the phrasal tags of the constituency tree (such as NP, ADJP, ADVP etc.). Then, it goes over the dependency relations to detect each terminal node. The output at this point is the terminal nodes and the relationship information between these nodes derived from the dependency tree. In the next step, links between terminal nodes are generated, thus subtrees are created. For a sentence like "Ali cesur kadından yardım aldı." (lit. "Ali brave woman.ABL help take.PAST.3pSING", *"Ali got help from the brave woman."*) the algorithm starts with identifying leaves and their immediate nodes:

Ali	cesur	kadından	yardım	aldı.
Ali.NP	brave.ADJ	woman.NP	help.NP	get.VP

Then the algorithm proceeds to create subsubtrees.

First the VP[NP[yardım] VP[aldı]] subtree (See Figure 6) is created as the hierarchy of subtree appendage suggests (see Figure 5), then NP[ADJP[cesur] NP[kadından]] (See Figure 7) is created since it has no child nodes:

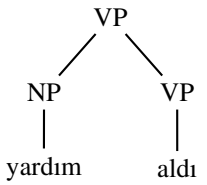

Figure 6: "yardım aldı" subtree

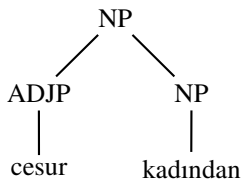

Figure 7: "cesur kadından" subtree

As the dependency tag of [NP[Ali]] is NSUBJ, it is directly linked to the S node. That is why it is excluded

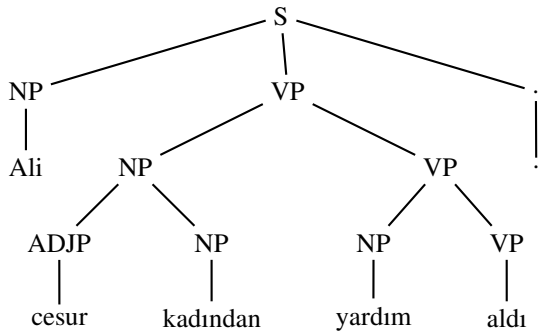

Figure 8: "Ali cesur kadından yardım aldı."

```
NOUN CONJ NOUN 0 2 NMOD 1 2 CC 1
NOUN VERB PUNC 0 1 NSUBJ 2 1 PUNCT 1
NOUN NOUN NOUN 0 2 NMOD 1 2 NMOD 1
NOUN NUM NOUN 0 2 NMOD 1 2 NUMMOD 3
NOUN ADV POSTP 0 1 NMOD 2 1 CASE 2
```

Figure 9: Sample set of instances

Prior to changes:

```
ADJ ADJ NOUN 0 2 AMOD 1 2 AMOD 1
```

After changes:

```
ADJ NOUN false false false true false
false ADJ ADJ false false false false
false false NOUN NOUN false false false
false false false 0 2 AMOD 1 2 AMOD 1
```

Figure 10: Sample ADJ, ADJ, NOUN instance

from this step. For the third step, subtrees and childless nodes are merged and linked to the root S, thus creating the final tree in Figure 8. One of the most important features of the algorithm is that it first detects the subject NP and predicate VP using NSUBJ tag and ROOT tag from the dependency structure. Then everything on the left hand side of the subject NP head is considered as part of subject NP. Everything else except the (.) is considered to be a child of the predicate VP. (.) is directly connected to the S node. The algorithm strictly follows this linguistic principle while merging subtrees.

3.2.2. Classifier Oracle

A classifier oracle is employed for the machine learning model. The defining characteristic of the classifier oracle is that it merges the subtrees in accordance with their class information.

This class information is provided by the predetermined classes based on the number of dependents each head has. There are four classes for heads: 1-dependent, 2-dependent, 3-dependent and 4-dependent. The constraints and features regarding these classes are determined making use of dependency tags, POS tags and syntactic particularities of Turkish.

The algorithm follows a bottom-up processing style: It starts from the terminal nodes and builds the constituency structure up until it reaches the root, which is S. Similar to the basic oracle, classifier oracle refers to the morphological analyser to pull basic tags and create phrasal tags like ADJP, NP and VP. Then, dependency relationships between the leaves of the subtree are referred to (See Figure 11).

Length	Error Rate
3	21.06%
4	17.35%
5	17.35%
6	10.4%
7	4.79%
8	1.8%

Table 1: Error rates of the subtrees according to their length (Bagging, Ensemble Size = 200)

In order to create a trainable model, a series of instances that consisted of Left, Right and Merge commands are created (See Figure 11). Each of these instances carries information about the group to be merged, including the POS-tags of all of the words in the group, indices of each word in the group, indices of the words they connect to, and their dependency relations (See Figure 9).

An instance for a subtree that consists of three nodes whose POS-tags are NOUN, VERB, and PUNC respectively, will carry the POS-tag information for each node, the index of the first connecting node (it is 0 here for NOUN), index of the node that it connects to (it is 1 here for VERB), and the dependency relation (here it is NSUBJ). After nodes are connected in accordance with headedness and information listed above, the algorithm moves on to indicating other possible connections in the group. In this example, next step is to indicate 2 for the connecting node PUNC, 1 for the target node VERB and PUNCT for the name of the flag, and finally in order to indicate the class of the group, index of the head is stated, in this example being 1 for the verb head.

After these instances (Left-Right-Merge, Left-Merge-Right-Merge and Right-Merge-Left-Merge) are created, they are sorted in accordance with the nodes that requires merging. Then, they are turned into class information which allows training the model using more efficient commands.

3.2.3. Training and Testing the Model

In order to find the most efficient model for the purposes of this study, different machine learning algorithms are tested: Decision Tree, Naive Bayes, KNN, Random Forest, and Bagging. Bagging performed better than the other alternatives yielding less errors (See Table 1). Thus, it was decided to proceed with this model.

In addition, an experimental run was carried out where features of the left and right context were referred to.

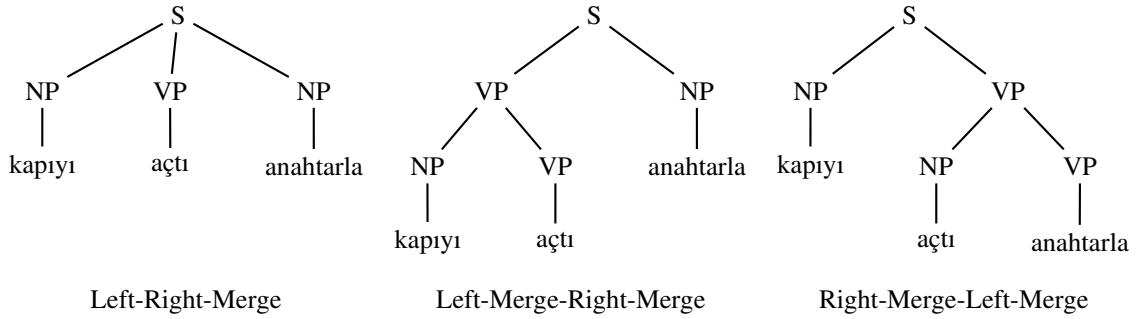

Figure 11: "[S/he] opened the door with a key" 3 merging possibilities of 3 leaf nodes. Each one will correspond to one class. "açtı'" (lit. "open.PAST.3PSING", *"opened"*) is the head node for this example.

When the features of two preceding words and two following words were incorporated into computation, error rates increased due to added complexity. Hence this approach was abandoned quickly.

After settling on a model, few alternations to the features were tested to attain better results. Replacing some features with others and deleting the existing ones did not yield better results. Thus, new features were added to improve performance. The final version of the algorithm refers to the POS tag of each leaf node and their morphological analysis which offers information on case inflection (ablative, dative, genitive, nominative, accusative, instrumental) and whether the lemma is a proper noun. Hence, the algorithm first checks the morphological information and POS tags associated with the words. For a subtree that consists of ADJ, ADJ, and NOUN, it extracts these tags along with the morphological information. If the root bears the inflection the algorithm checks for, it is assigned a *true* value; if the root does not bear that specific inflection, it is assigned *false* value. This operation is iterated for each inflection and word. This creates a sequence of binary *true/false* outputs for the instance (See Figure 10).

Here in Figure 10, ADJ is the POS tag of the first word and NOUN is the POS tag of its root word. The first three false values refer to the fact that the word does not bear ablative, dative nor genitive inflection. The *true* value indicates that this word bears nominative inflection. The remaining false values indicate that the word does not bear accusative inflection and is not a proper noun.

After morphological information is processed by the algorithm, it moves on to checking the indices of each head and their dependents. Then it extracts dependency relation AMOD, and finally, it yields the index of the head word.

4. Evaluation Process

Studies on conversion have different stances on evaluating the performance of conversion algorithms: Some scholars opt for sharing the F-score, recall and precision of their algorithm (Bick, 2006; Xia and Palmer, 2001; Lee and Wang, 2016) while others skip this step (Lu et al., 2016; Bosco, 2007). As this study offers the very first conversion algorithm for Turkish dependency trees, a careful evaluation of the algorithm's performance was conducted in order to set a baseline for further studies. F-scores of the baseline (See Table 2) and classifier oracle (See Table 3) can be found in this section. For a thorough discussion of the evaluation process and evaluation metric, see 4.1.

Table 2: Precision, recall and F-score values of basic oracle

Precision	Recall	F-Score
88.51	89.90	89.20

Table 3: Precision, recall and F-score values of classifier oracle

Precision	Recall	F-Score
89.34	90.06	89.70

4.1. Evaluation Metric

After converting dependency trees into constituency trees using the algorithm discussed in 3.2, the evaluation process started. Since parser evaluation is still a heated point of discussion, various different metrics are employed: Head attachment score, label precision, labelled attachment score, branch precision, GEIG, PARSEVAL (Black et al., 1991) and Leaf-Ancestor (Sampson and Babarczy, 2002). Since our algorithm employs morphological analyser to create tags for each leaf, incorporating metrics like label precision and label attachment score would measure the accuracy of the morphological analyser instead of the conversion algorithm. That is why PARSEVAL and Leaf-Ancestor were the top competitors for being the evaluation metric. PARSEVAL is considered to be the canonical metric. It has been criticised heavily for failing to capture the true performance of parsers (Sampson and Babarczy, 2002) mainly because it has a tendency to punish parsers that make more mistakes while attempting to offer more information by making more claims. Moreover, it is more coarse compared to

Table 4: Confusion matrix of the tags

Tag	ADJP	ADVP	CONJP	DP	INTJP	NEG	NOMP	NP	NUM	PP	S	VP	WP
ADJP	**92.2**	2.0	0	0.3	0	0	0.1	2	0.1	0.2	0	0.8	0
ADVP	2.6	**71.8**	0.3	0.1	0.1	0	0.2	21.2	0.1	0.8	0	1.4	0
CONJP	0	8.4	**89.2**	0	0.1	0	0	0.3	0	0.1	0	0.3	0
DP	2.7	0.3	0	**94.7**	0	0	0	0.2	1.4	0	0	0	0
INTJP	0	10.0	0	0	**80**	0	0	3.3	0	0	0	3.3	0
NEG	0.4	0	0	0	0	**99.1**	0	0	0	0	0	0.4	0
NOMP	14.9	1.8	0	0.3	0	0	**27.5**	21.0	0.4	0.5	0.1	21.5	0
NP	0.7	0.2	0	0.3	0	0	0.2	**93.2**	0.1	0.5	0.2	0.5	0
NUM	0.1	0	0	1.2	0	0	0	3.9	**92.8**	0.1	0	0	0
PP	0.1	2.4	0.6	0	0	0	0.1	4.6	0.1	**89.3**	0	0.4	0
S	0.3	0	0	0	0	0	0.1	0.2	0	0	**96.8**	0.8	0
VP	0.3	0	0	0	0	0	0.2	0.2	0	0.1	0.1	**90.9**	0
WP	1.2	8.6	2.5	0	0	0	0	3.7	0	0	0	1.2	**82.7**

Leaf-Ancestor metric since the former compares brackets and later compares the path from leaf node to root node using Levenshtein distance. Since our intention was not introducing the accuracy of POS tags as a metric, we opted out of using Leaf-Ancestor.

Our algorithm created flatter trees, thus we intended to shift the focus of our evaluation to brackets and incorporation of subtrees. Also our constituency trees only had basic tags like NP rather than function specifying ones like NP-Subj, NP-Obj and such. Consequently, PARSEVAL metric was a better fit than Leaf-Ancestor for us due to numerous reasons including the fact that it weighs all nodes evenly instead of focusing on certain syntagmatic and/or semantic relations more. On the other hand, PARSEVAL failed to assess the subtree grouping performance of our algorithm as it processed parse trees in a bottom-up fashion and it does not consider phrase tags while assessing similarity. That is why we came up with a new evaluation algorithm which operates in a top down fashion.

After conversion process was concluded, a set of linguists manually annotated constituency trees of the dependency structures used as the input of the conversion algorithm. These manually annotated constituency trees served as the gold standard in parser evaluation process. In order to assess the performance of the conversion algorithm, the evaluation algorithm starts comparing output trees and gold standard trees from the top node. It compares the phrase below each node of the conversion output and gold standard until it reaches the individual leaves. Anything other than exact matches are penalized.

4.2. Results

The overall performance of the rule based algorithm is satisfactory with an F-score of 89.20 but it is not as successful as the machine learning algorithm which has an F-score of 89.70 (See Table 3 for details). The success of morphological analyser (Yıldız et al., 2019) and carefully created the hierarchy of subtree appendage (Figure 5) are the main contributors of these F-score, precision and recall values. In addition, the relatively shallow structure of the constituency trees offer less information compared to trees that use bar levels and more sophisticated tags that also show semantic roles. As a result, precision and recall values of the uncomplicated and clearcut trees are quite high.

Accuracy of the NEG tag shows the peak performance of the rule-based algorithm (Table 4) as Turkish negation of nominal and copular predicates is a distinct word bearing only the meaning of negation, this result is expected. Accuracy of S tag follows it with 96.8. Since the information to detect the highest node S is encapsulated in the ROOT tag of the dependency, this performance may seem expected yet bearing in mind the fact that predicates of embedded sentences are also tagged with S, a 96.8 accuracy value is very impressive.

Accuracy values of WP (82), INTJP (80), ADVP (71.8), and NOMP (27.5) are the lowest (Table 4). The morphologically rich typology of Turkish allows deriving adjectives from verbal roots and stems. Such adjectivals lead to a small degree of fuzz in the data. However, detection of ADJPs still do not perform badly with 92.2. Moreover, some adjectives can also be used as determiners in certain cases. This quirk of Turkish drops the accuracy of DP. Similarly, some conjunctions can be used as interjections, hence the significantly low precision of INTJPs and some adjectives and adverbs have the same word form in Turkish which drops the score of ADVP. As a newly introduced tag, NOMP is as low as 27.5 the reason why, from a purely linguistic perspective, is its unpredictable environment. With the introduction of our machine learning model, performance of NOMP gets higher to 36.

Overall, the greatest confusion was caused by the vastness of the nominal domain in Turkish. The distinction between nominals and verbs is pretty clearcut while distinguishing nouns from adverbs and adjectives is rather a challenging task. That is why the morphologic analyser and consequently the conversion algorithm had the most trouble with this task.

As anticipated, performance of machine learning algorithm is overall better. Accuracy of the NEG tag is the highest (99.1) followed by the S tag which has 96.9

score. Accuracy of NOMP tag is the lowest with machine learning algorithm as well; however, we see significant improvement as its score is now 36.

5. Conclusion

In conclusion, the algorithm developed in this study creates flatter trees using a machine learning based model, predefined set of heuristics, and morphology-based tags with high accuracy. A total of 8786 trees were created. The overall constituency structures were flatter due to the following reasons:

- Not employing the bar-level representations,
- Choosing phrasal tags based on the POS tag of the head of the each phrase,
- Not including empty projections helped making the overall structure flatter.

Therefore, the conversion algorithm's output offered simpler trees with higher accuracy rates. The aim of implementing rules like tag hierarchy and no bar level representations was being able to offer a linguistically sound constituency structure that fitted typology of the Turkish language. Movement, co-indexation and empty projections were not illustrated since Turkish licenses free word order and scrambling. As a result, a single sentence can have many variations depending on the focus position and emphasis.Thus, the movement and co-indexation was not essential to be included. Other than that, the constituency trees created by the algorithm reflect the syntactic features of Turkish as they illustrate particularities like nominal predicates.

The main goal of this study was setting a baseline for dependency to constituency conversion algorithm. Offering two different methods (rule-based and machine learning-based) with high success rates, we believe that we fulfilled this goal. Yet different machine learning models and neural networks can be tested out in the future studies to increase the performance of the conversion algorithm even more.

6. Bibliographical References

Bai, J., Wang, Y., Chen, Y., Yang, Y., Bai, J., Yu, J., and Tong, Y. (2021). Syntax-bert: Improving pre-trained transformers with syntax trees. arXiv.

Bick, E. (2006). Turning a dependency treebank into a PSG-style constituent treebank. In *Proceedings of the Fifth International Conference on Language Resources and Evaluation (LREC'06)*, Genoa, Italy, May. European Language Resources Association (ELRA).

Black, E., Abney, S., Flickenger, D., Gdaniec, C., Grishman, R., Harrison, P., Hindle, D., Ingria, R., Jelinek, F., Klavans, J., Liberman, M., Marcus, M., Roukos, S., Santorini, B., and Strzalkowski, T. (1991). A procedure for quantitatively comparing the syntactic coverage of English grammars. In *Speech and Natural Language: Proceedings of a Workshop Held at Pacific Grove, California, February 19-22, 1991*.

Bosco, C. (2007). Multiple-step treebank conversion: From dependency to Penn format. In *Proceedings of the Linguistic Annotation Workshop*, pages 164–167, Prague, Czech Republic, June. Association for Computational Linguistics.

Bouma, G. and Kloosterman, G. (2002). Querying dependency treebanks in XML. In *Proceedings of the Third International Conference on Language Resources and Evaluation (LREC'02)*, Las Palmas, Canary Islands - Spain, May. European Language Resources Association (ELRA).

Cahill, A., Burke, M., O'Donovan, R., Riezler, S., van Genabith, J., and Way, A. (2008). Wide-coverage deep statistical parsing using automatic dependency structure annotation. *Computational Linguistics*, 34(1):81–124.

Candito, M., Crabbé, B., and Denis, P. (2010). Statistical French dependency parsing: Treebank conversion and first results. In *Proceedings of the Seventh International Conference on Language Resources and Evaluation (LREC'10)*, Valletta, Malta, May. European Language Resources Association (ELRA).

Chomsky, N. (1957). Syntactic structures (the hague: Mouton, 1957). *Review of Verbal Behavior by BF Skinner, Language*, 35:26–58.

Chomsky, N. (1968). *Remarks on Nominalization*. Linguistics Club, Indiana University.

Daum, M., Foth, K., and Menzel, W. (2004). Automatic transformation of phrase treebanks to dependency trees. 06.

Ding, Y. and Palmer, M. (2004). Synchronous dependency insertion grammars: A grammar formalism for syntax based statistical MT. In *Proceedings of the Workshop on Recent Advances in Dependency Grammar*, pages 90–97, Geneva, Switzerland, aug 28. COLING.

Ding, Y. and Palmer, M. (2005). Machine translation using probabilistic synchronous dependency insertion grammars. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL'05)*, pages 541–548, Ann Arbor, Michigan, June. Association for Computational Linguistics.

Grammar, C. and Hockenmaier, J. (2003). Data and models for statistical parsing with combinatory categorial grammar. 09.

Hajic, J., Brill, E., Collins, M., Hladká, B., Jones, D., Kuo, C., Ramshaw, L., Schwartz, O., Tillmann, C., and Zeman, D. (1998). Core natural language processing technology applicable to multiple languages–workshop'98. Technical report, Technical report, Johns Hopkins Univ.

Höfler, S. (2002). Link2tree: A dependency-constituency converter. 01.

Huang, L., Knight, K., and Joshi, A. (2006). Statistical

syntax-directed translation with extended domain of locality. 01.

Kara, N., Aslan, D. B., Marşan, B., Bakay, Ö., Ak, K., and Yıldız, O. T. (2020). TRopBank: Turkish PropBank v2.0. In *Proceedings of the 12th Language Resources and Evaluation Conference*, pages 2763–2772, Marseille, France, May. European Language Resources Association.

Koehn, P., Och, F. J., and Marcu, D. (2003). Statistical phrase-based translation. *University Of Southern California Marina Del Rey Information Sciences Inst*.

Kong, L., Rush, A. M., and Smith, N. A. (2015). Transforming dependencies into phrase structures. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 788–798, Denver, Colorado, May. Association for Computational Linguistics.

Lee, Y.-S. and Wang, Z. (2016). Language independent dependency to constituent tree conversion. In *COLING*.

Lu, A., Malamud, S., and Xue, N. (2016). Converting syntagrus dependency treebank into penn treebank style. pages 16–21, 01.

Matthews, P. H. (1981). Syntax. cambridge etc. Cambridge University Press (Cambridge Textbooks in Linguistics). Indices.

Nivre, J., de Marneffe, M.-C., Ginter, F., Hajič, J., Manning, C. D., Pyysalo, S., Schuster, S., Tyers, F., and Zeman, D. (2020). Universal dependencies v2: An evergrowing multilingual treebank collection. *arXiv preprint arXiv:2004.10643*.

Nivre, J. (2003). Theory-supporting treebanks. In *Proceedings of the 2nd Workshop on Treebanks and Linguistic Theories*, pages 117–128.

Percival, W. K. (1990). Reflections on the history of dependency notions in linguistics. *Historiographia linguistica*, 17(1-2):29–47.

Sag, I., Wasow, T., and Bender, E. (2003). Syntactic theory: A formal introduction, 2nd edition. *Bibliovault OAI Repository, the University of Chicago Press*, 01.

Sampson, G. and Babarczy, A. (2002). A test of the leaf-ancestor metric for parse accuracy. In *The Workshop Programme*, page 23.

Shen, L., Xu, J., and Weischedel, R. (2008). A new string-to-dependency machine translation algorithm with a target dependency language model. In *Proceedings of ACL-08: HLT*, pages 577–585, Columbus, Ohio, June. Association for Computational Linguistics.

Wang, W. and Harper, M. P. (2004). A statistical constraint dependency grammar (cdg) parser. In *Proceedings of the Workshop on Incremental Parsing: Bringing Engineering and Cognition Together*, IncrementParsing '04, pages 42–49, USA. Association for Computational Linguistics.

Xia, F. and Palmer, M. (2001). Converting dependency structures to phrase structures. In *Proceedings of the First International Conference on Human Language Technology Research*.

Xie, J., Mi, H., and Liu, Q. (2011). A novel dependency-to-string model for statistical machine translation. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing*, pages 216–226, Edinburgh, Scotland, UK., July. Association for Computational Linguistics.

Yıldız, O. T., Avar, B., and Ercan, G. (2019). An open, extendible, and fast Turkish morphological analyzer. In *Proceedings of the International Conference on Recent Advances in Natural Language Processing (RANLP 2019)*, pages 1364–1372, Varna, Bulgaria, September. INCOMA Ltd.