Assignment 2: Transition-Based Dependency Parsing

Course: Linguistic Data 3: Data Modeling in ILs

Student Name: Vivek Hruday Kavuri

Roll Number: 2022114012

1. Introduction

The objective of this assignment was to implement and analyze a transition-based dependency parser. Dependency parsing is a method of syntactic analysis that identifies binary relationships (dependencies) between words in a sentence, where one word is the head (governor) and the other is the dependent (modifier).

This report details the implementation of the **Arc-Eager** transition system, the associated **Static Oracle**, and the evaluation of the parser on the **Hindi Treebank (HUTB)**. Furthermore, as an extra credit task, the **Arc-Standard** transition system was implemented and compared against the Arc-Eager baseline.

2. Implementation Details

2.1 The Arc-Eager Transition System

The Arc-Eager system is a transition-based parsing algorithm that builds a dependency tree incrementally. It utilizes a **Stack** (containing partially processed words) and a **Buffer** (containing input words). The system defines four transitions:

1. **Shift:** Moves the first word from the buffer to the stack.
2. **Left-Arc:** Creates a dependency arc from the buffer front (head) to the stack top (dependent) and pops the stack.
    - *Precondition:* Stack is not empty; the token on top of the stack does not already have a head.
3. **Right-Arc:** Creates a dependency arc from the stack top (head) to the buffer front (dependent) and shifts the buffer front to the stack.
    - *Precondition:* Stack is not empty.
4. **Reduce:** Pops the stack.
    - *Precondition:* The top of the stack already has a head.

This system is "eager" because it creates arcs as soon as the head and dependent are available, specifically allowing right-dependents to be attached before their own dependents are processed.

2.2 The Static Oracle

To train the parser (or in this assignment's case, to simulate perfect parsing), a Static Oracle was implemented. The Oracle determines the correct transition to take given the current configuration and the Gold Standard tree. The logic implemented is as follows:

1. **Left-Arc Condition:** If the head of the stack top is the buffer front.
2. **Right-Arc Condition:** If the head of the buffer front is the stack top.
3. **Reduce Condition:** If the stack top has already been assigned a head, and there are no more dependents for the stack top remaining in the buffer.
4. **Shift Condition:** Default action if no other conditions are met.

2.3 Data Preprocessing (Hindi Treebank)

The assignment required testing on real-world data using the Hindi Treebank. A preprocessing script (convert_hindi_to_tab.py) was utilized to convert the raw Hindi data into the CoNLL-X tab-separated format (.tab) required by the parser.

- **Input:** Hindi Treebank files.
- **Output:** hindi_dev.tab and hindi_test.tab.

3. Experimental Setup & Results

3.1 Evaluation Metrics

The parser was evaluated using two standard metrics:

- **UAS (Unlabeled Attachment Score):** The percentage of words that have been assigned the correct head.
- **LAS (Labeled Attachment Score):** The percentage of words that have been assigned both the correct head and the correct dependency label.

3.2 Arc-Eager Performance (Baseline)

The Arc-Eager system was run on the Hindi Development and Test sets.

Dataset	Total Tokens	UAS (Unlabeled)	LAS (Labeled)
Dev Set	37,736	**97.96%**	**97.80%**
Test Set	38,954	**98.07%**	**97.91%**

The system achieved very high accuracy (>97%), indicating that the Oracle logic correctly covers the vast majority of syntactic structures present in the Hindi Treebank.

4. Error Analysis

Despite high accuracy, specific mismatches were observed in the output logs. Analyzing these discrepancies provides insight into the limitations of the transition system or the oracle's handling of specific linguistic phenomena.

4.1 Case Study 1: Relative Clause Modifiers (nmod__relc)

Sentence: "... *murti ki khoj ki*" (search for the idol was done)

- **Token:** ki (Genitive marker/Verb auxiliary context)
- **Gold Head:** 1 (relational modifier)
- **Predicted Head:** 0 (Root)
- **Analysis:** The label nmod__relc denotes a relative clause modifier. The mismatch here shows the parser defaulting to root. In Arc-Eager, if the algorithm "Reduces" too early or fails to attach a Right-Arc because of complex intervening clauses, the unattached token often remains on the stack or buffer until the end, defaulting to Root. This suggests an issue with resolving long-distance dependencies where the head is far removed from the dependent.

4.2 Case Study 2: Punctuation (rsym)

Sentence: "... *khoj ki |*" (end of sentence marker)

- **Token:** | (Danda/Full stop)
- **Gold Head:** 10
- **Predicted Head:** 0 (Root)
- **Analysis:** Punctuation attachment is a common source of error. In dependency grammar, punctuation usually attaches to the main verb of the clause. If the parser fails to identify the main verb correctly (or Reduces it too early), the punctuation is left stranded and defaults to the Root attachment.

4.3 Case Study 3: Locative/Spatial Arguments (k7p)

Sentence: "... *aur kedarnath mein* ..."

- **Token:** aur (and)
- **Gold Label:** k7p (Location)
- **Predicted Label:** root
- **Analysis:** This error occurred in the Test set. The token aur is a conjunction. In Hindi dependency standards (Panini), conjunctions can be tricky. Here, the gold standard treats it as part of a locative phrase, while the parser failed to find the attachment. This highlights the difficulty transition systems have with **Coordination**, which often requires look-ahead capabilities that standard Arc-Eager lacks.

5. Extra Credit: Arc-Standard Transition System

For the extra credit component, I implemented the **Arc-Standard** transition system.

5.1 Implementation Difference

Unlike Arc-Eager, Arc-Standard is a "bottom-up" approach. It does not have a Reduce transition. Instead:

1. **Left-Arc:** Head is on Buffer, Dependent is on Stack. Pop Stack.
2. **Right-Arc:** Head is on Stack, Dependent is on Buffer. **Shift Buffer head to Stack** (unlike Eager), create arc, and eventually pop dependent when it is on top of the stack and head is second.
3. **Constraint:** In Arc-Standard, a Right-Arc is only formed when the dependent has collected *all* of its own dependents.

5.2 Comparative Results

System	Dataset	UAS	LAS
Arc-Eager	Test Set	**98.07%**	**97.91%**
Arc-Standard	Test Set	97.54%	97.54%

5.3 Discussion

The Arc-Eager system slightly outperformed the Arc-Standard system on this Hindi dataset (approx. +0.5%).

- **Arc-Eager Advantage:** Arc-Eager allows for right-attachments to be made immediately (before the dependent finds its own dependents). This "eager" behavior can be beneficial for ensuring arcs are created early, reducing the size of the stack and potentially simplifying state management for the Oracle.
- **Arc-Standard Limitation:** Because Arc-Standard must wait until a subtree is fully complete before attaching it to its head (strictly bottom-up), it can sometimes require a larger lookahead or stack depth to resolve dependencies correctly, making the Oracle's job slightly more complex in cases of deep nesting.

6. Conclusion

In this assignment, I successfully implemented a transition-based dependency parser for Hindi. The Arc-Eager system proved to be highly effective, achieving >97.9% labeled accuracy on the test set. The error analysis revealed that residual errors are largely concentrated in complex structures like relative clauses, coordination, and punctuation attachment. The comparison with Arc-Standard demonstrated that while both systems are effective, the eager attachment strategy offered a slight performance edge for this specific dataset and configuration.

7. Github Repository

https://github.com/vivekhruday05/LD3-S-26-Assignments/tree/main/2