Lab 5X: Projectivization

Marco Kuhlmann

Goal	Implement (a variant of) the Eisner algorithm and apply it to a practical problem in dependency parsing.
Preparations	Watch the video lecture on the Eisner algorithm (Lecture 5.3).
Report	Submit the requested code, as well as a short reflection piece (ca. 150 words) about your experience.
Deadline	2023-02-24

Introduction

Many algorithms for syntactic dependency parsing, including the Eisner algorithm and the arc-standard algorithm, are restricted to projective dependency trees. At the same time, several dependency treebanks contain non-projective trees dependency trees. A common solution to this problem is to transform the (possibly non-projective) trees of the original treebank into projective trees prior to training – a transformation known as *projectivization*. Your task in this practical is to implement a projectivizer based on the Eisner algorithm.

Background

When projectivizing a dependency tree, it is desirable to maintain as much of the structure of the original tree as possible. More specifically, linguistic considerations tell us that it makes sense to only allow changes to the source tree that move the head of a dependency arc $h \rightarrow d$ to one of the *ancestors* a of h. This is known as *lifting*. Applying lifting to a dependency tree makes the tree 'more' projective, in the sense that the number of crossing arcs in the tree at most decreases (compared to the original tree). In the extreme case, all arcs of the source tree can be lifted to the root

vertex, in which case the target tree is guaranteed to be projective. More generally though, we are interested in computing a projective dependency tree that requires a *minimal* amount of lifting. One can prove that, for every dependency tree, there is a unique projective dependency tree that satisfies the minimal lifting property.

To compute a projective dependency tree that requires a minimal amount of lifting, we can use a variant of the Eisner algorithm. Given an arc $h \rightarrow d$ from a head vertex h to a dependent vertex d in the source tree, we define the *cost* of a potential arc $h' \rightarrow d$ in the target tree as the number of lifting operations that it takes to re-attach d to h', or positive infinity in case this is not possible, that is, when h' is not an ancestor of h in the source tree. Note that the cost is zero for arcs that are simply copied from the source tree to the target tree, that is, where $h' = h$. More generally, the cost is equal to the length of the (unique) path in the source tree from h' to h, in the case where such a path exists. We can then adapt the Eisner algorithm to find a projective dependency tree with *minimal cost*, rather than maximal score.

Problem specification

You will have to write a script that reads in dependency trees in the CoNLL-U format, for each tree computes a projective dependency tree that minimizes the total amount of lifting as explained above, and outputs the transformed tree. To achieve this goal you will have to implement the Eisner algorithm, including a backpointer mechanism for constructing the dependency tree with minimal lifting. Your script should work in the same way as the script `projectivize.py`, which is linked from the Project page (D3: Baseline system). That script implements another algorithm for projectivization; however, both algorithms produce the same results. To test your implementation, you can therefore compare the output of your script to the output of `projectivize.py` using a tool such as `diff`; the two outputs should be identical.