Phrase-Projection for Cross-Lingual JAMR training

Adithya Renduchintala

September 4, 2014

1 Introduction

We are given spans of the target text which align to concepts in the AMR graph.These alignment do not cover every token in the target sentence. Typically function words are not aligned to any graph fragment. Next, we obtain word alignments between the target sentence and source sentence. Since we have word alignments between target and source, and phrase alignments between target and AMR graph, we must convert the word alignments into phrase alignments. The phrases on the source side will then be projected to the AMR concepts via the target sentence. When obtaining the phrases on the source side, two constrains that we are enforcing are:

- Every phrase/span in the target side that is associated with a concept/graph fragment there should be a phrase/span in the source side
- The spans on the source side should be non-overlapping.

Below are diagrams showing phrases suitable for projection, and one phrase alignment that is not suitable.

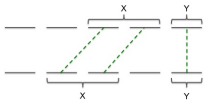

Figure 1: Acceptable Phrase Projection

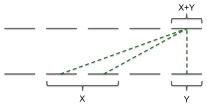

Figure 2: Acceptable Phrase Projection

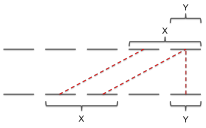

Figure 3: Violating Phrase Projection

2 Model

In our scenario the source sentence is Chinese and the target is English.

$$F = [f_1, f_2, f_3, ...f_m]$$
$$E = [e_1, e_2, e_3, ..e_n]$$

We run JAMRs rule based alignment tool to obtain alignments between the English sentence and the AMR graph. These are phrase to subgraph alignments. These phrases are taken as a given segmentation of the target.

$$P_E = [p_{T1}, p_{T2}, ...p_{Tk}]$$

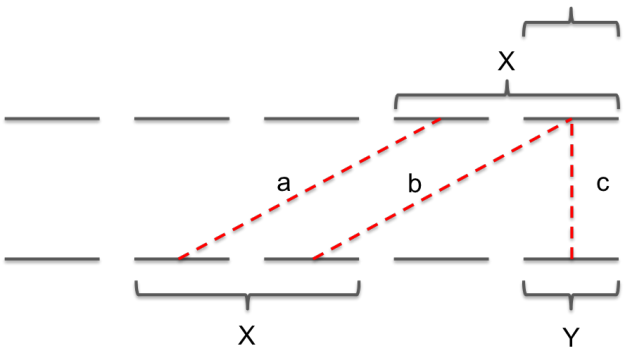

Figure 4: An example of phrase overlap

Where p_i is the i^{th} phrase in the sentence. We want to find a segmentation and an alignment between phrases that maximizes the phrase alignments between the two sentences.

$$P_F = [p_{F1}, p_{F2}, ...p_{Fk}]$$
$$A = [a_1, a_2, a_3, ...a_n]$$

Given a phrase in the target p_{Ti} we know that each token in it has alignments $\{a_i, a_{i+1}, ...a_{i+j}\}$, this phrase has $j+1$ tokens. We define the corresponding phrase p_{Fi} as the span created between $\min\{a_i, a_{i+1}, ...a_{i+j}\}$ and $\max\{a_i, a_{i+1}, ...a_{i+j}\}$.

$$p_{Fi} = \min a_i, a_{i+1}, ...a_{i+j}, \max a_i, a_{i+1}, ...a_{i+j} \tag{1}$$

In this manner we can define all the spans in F as long as the two constraints defined are not violated. However, if the constraints are violated then we have to change the alignments forming the spans. The

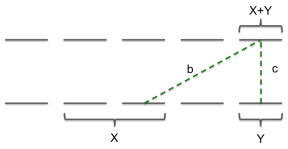

Figure 5: removed alignment a

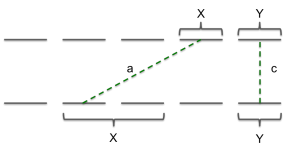

Figure 6: removed alignment b

example of a violation shown in 4 can be corrected by dropping some subset of alignment edges.
For example we can arrive at two solutions by dropping either the alignment edges a or b (5, 6). The alignment edge c can not be dropped because it mean that an entire phrase in the target can not be projected over to the source. The best alignment and segmentation can be obtained by using:

$$= \underset{\mathcal{A},\mathcal{K}}{\operatorname{argmax}} \prod_{i}^{|K|} P(P_{Fa_i} \mid P_{Ti}) \tag{2}$$

Where $\mathcal{A} \subset A$ that ensures that the constraints are met.

3 Algorithm

One exponential time algorithm to solve the task is presented below. The algorithm is essential a Breath-First Tree search algorithm. Each node in the tree represents a set of alignments. These sets are all subset of the full set of word alignments A. The search process starts with checking if source phrases can be created using the using expression 1.
At each node, one alignment is removed and the set formed by removing one alignment forms a child node. During the search, each node (set of alignments) is checked for complying with the constraints. If the node is conforming, then the list of alignments is a solution. The CheckAlignment method checks for overlapping phrases on the source side. This will make sure that only alignments which comply with the non-overlap constraint will be accepted in the final list of solutions.
The RemoveAlignment method removes an alignment link from the set of alignments which is used to create a child node. In this method, the if statement ensures that if an alignment is the only alignment for a target phrase then it will not be removed. This make sure phrases in the target will have a projection in the source.

3.1 Complexity

This is an exponential algorithm, each step in the search opens a node in the graph. There is some amount of recombination but the middle layer of the tree will be very wide. The recombination happens because a node which has had the same alignments removed (in different orders) will be the same, i.e. different alignment removal orders can result in the same node. The fan out is exponential in the number of alignments.
In practice, however, most of the alignments are the only alignment from a target phrase. This is in turn because most of the target phrases are single words. This means most of the alignments can not be removed, this drastically reduces the search space. Only the small amount of multi-word phrases in the the target will have to be searched, if the alignments do not match the constraints.

Algorithm 1 Phrase Align

```
procedure CHECKALIGNMENT(alignments A)
    for phrase p_T in P_T do
        p_F = getSourcePhrase(p_T)
        for phrase p_t in getAssociatedPhrase(p_F) do
            if p_t != p_T then
                return False
            end if
        end for
    end for
    return True
end procedure
procedure REMOVEALIGNMENT(Alignments A, alignment a)
    if a from p_T of span 1 then
        return A                          ▷ Can not remove a single word phrase's alignment
    else
        return Â
    end if
end procedure
procedure SOLVEALIGNMENT(alignment A)
    final = []                                          ▷ Final list of alignments
    Q = []                                              ▷ Initialized Que
    Q.push(A)
    while Q not empty do
        A =Q.pop()
        if CheckAlignment(A) then
            final.add(A)
        else if
            for  dothenalignment a in A
                Â = RemoveAlignment(A,a)
                if A != Â then
                    pass                    ▷ Branch ends, could not remove any alignment
                else if
                    thenQ.push(Â)
                end if
            end for
        end if
    end while
    return final
end procedure
```