

Partial Parsing from Bitext Projections

Prashanth Mannem and Aswarth Dara

Language Technologies Research Center

International Institute of Information Technology

Hyderabad, AP, India - 500032

{prashanth, abhilash.d}@research.iiit.ac.in

Abstract

Recent work has shown how a parallel corpus can be leveraged to build syntactic parser for a target language by projecting automatic source parse onto the target sentence using word alignments. The projected target dependency parses are not always fully connected to be useful for training traditional dependency parsers. In this paper, we present a greedy non-directional parsing algorithm which doesn't need a fully connected parse and can learn from partial parses by utilizing available structural and syntactic information in them. Our parser achieved statistically significant improvements over a baseline system that trains on only fully connected parses for Bulgarian, Spanish and Hindi. It also gave a significant improvement over previously reported results for Bulgarian and set a benchmark for Hindi.

1 Introduction

Parallel corpora have been used to transfer information from source to target languages for Part-Of-Speech (POS) tagging, word sense disambiguation (Yarowsky et al., 2001), syntactic parsing (Hwa et al., 2005; Ganchev et al., 2009; Jiang and Liu, 2010) and machine translation (Koehn, 2005; Tiedemann, 2002). Analysis on the source sentences was induced onto the target sentence via projections across word aligned parallel corpora.

Equipped with a source language parser and a word alignment tool, parallel data can be used to build an automatic treebank for a target language. The parse trees given by the parser on the source sentences in the parallel data are projected onto the target sentence using the word alignments from the alignment tool. Due to the usage of automatic source parses, automatic word alignments and differences in the annotation schemes of source and

target languages, the projected parses are not always fully connected and can have edges missing (Hwa et al., 2005; Ganchev et al., 2009). Non-literal translations and divergences in the syntax of the two languages also lead to incomplete projected parse trees.

Figure 1 shows an English-Hindi parallel sentence with correct source parse, alignments and target dependency parse. For the same sentence, Figure 2 is a sample partial dependency parse projected using an automatic source parser on aligned text. This parse is not fully connected with the words *banaa*, *kottaige* and *dikhataa* left without any parents.

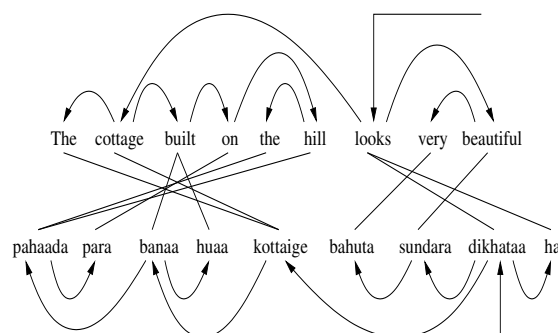


Figure 1: Word alignment with dependency parses for an English-Hindi parallel sentence

To train the traditional dependency parsers (Yamada and Matsumoto, 2003; Eisner, 1996; Nivre, 2003), the dependency parse has to satisfy four constraints: *connectedness*, *single-headedness*, *acyclicity* and *projectivity* (Kuhlmann and Nivre, 2006). Projectivity can be relaxed in some parsers (McDonald et al., 2005; Nivre, 2009). But these parsers can not directly be used to learn from partially connected parses (Hwa et al., 2005; Ganchev et al., 2009).

In the projected Hindi treebank (section 4) that was extracted from English-Hindi parallel text, only 5.9% of the sentences had full trees. In

Spanish and Bulgarian projected data extracted by Ganchev et al. (2009), the figures are 3.2% and 12.9% respectively. Learning from data with such high proportions of partially connected dependency parses requires special parsing algorithms which are not bound by *connectedness*. Its only during learning that the constraint doesn't satisfy. For a new sentence (i.e. during inference), the parser should output fully connected dependency tree.

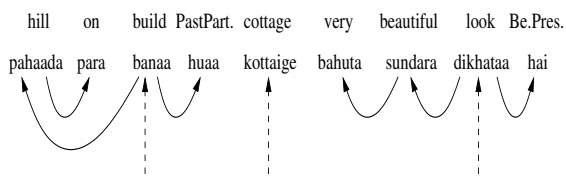


Figure 2: A sample dependency parse with partial parses

In this paper, we present a dependency parsing algorithm which can train on partial projected parses and can take rich syntactic information as features for learning. The parsing algorithm constructs the partial parses in a bottom-up manner by performing a greedy search over all possible relations and choosing the best one at each step without following either left-to-right or right-to-left traversal. The algorithm is inspired by earlier non-directional parsing works of Shen and Joshi (2008) and Goldberg and Elhadad (2010). We also propose an extended partial parsing algorithm that can learn from partial parses whose yields are partially contiguous.

Apart from bitext projections, this work can be extended to other cases where learning from partial structures is required. For example, while bootstrapping parsers high confidence parses are extracted and trained upon (Steedman et al., 2003; Reichart and Rappoport, 2007). In cases where these parses are few, learning from partial parses might be beneficial.

We train our parser on projected Hindi, Bulgarian and Spanish treebanks and show statistically significant improvements in accuracies between training on fully connected trees and learning from partial parses.

2 Related Work

Learning from partial parses has been dealt in different ways in the literature. Hwa et al. (2005) used post-projection completion/transformation

rules to get full parse trees from the projections and train Collin's parser (Collins, 1999) on them. Ganchev et al. (2009) handle partial projected parses by avoiding committing to entire projected tree during training. The posterior regularization based framework constrains the projected syntactic relations to hold approximately and only in expectation. Jiang and Liu (2010) refer to alignment matrix and a dynamic programming search algorithm to obtain better projected dependency trees. They deal with partial projections by breaking down the projected parse into a set of edges and training on the set of projected relations rather than on trees.

While Hwa et al. (2005) requires full projected parses to train their parser, Ganchev et al. (2009) and Jiang and Liu (2010) can learn from partially projected trees. However, the discriminative training in (Ganchev et al., 2009) doesn't allow for richer syntactic context and it doesn't learn from all the relations in the partial dependency parse. By treating each relation in the projected dependency data independently as a classification instance for parsing, Jiang and Liu (2010) sacrifice the context of the relations such as global structural context, neighboring relations that are crucial for dependency analysis. Due to this, they report that the parser suffers from local optimization during training.

The parser proposed in this work (section 3) learns from partial trees by using the available structural information in it and also in neighboring partial parses. We evaluated our system (section 5) on Bulgarian and Spanish projected dependency data used in (Ganchev et al., 2009) for comparison. The same could not be carried out for Chinese (which was the language (Jiang and Liu, 2010) worked on) due to the unavailability of projected data used in their work. Comparison with the traditional dependency parsers (McDonald et al., 2005; Yamada and Matsumoto, 2003; Nivre, 2003; Goldberg and Elhadad, 2010) which train on complete dependency parsers is out of the scope of this work.

3 Partial Parsing

A standard dependency graph satisfies four graph constraints: *connectedness*, *single-headedness*, *acyclicity* and *projectivity* (Kuhlmann and Nivre, 2006). In our work, we assume the dependency graph for a sentence only satisfies the single-

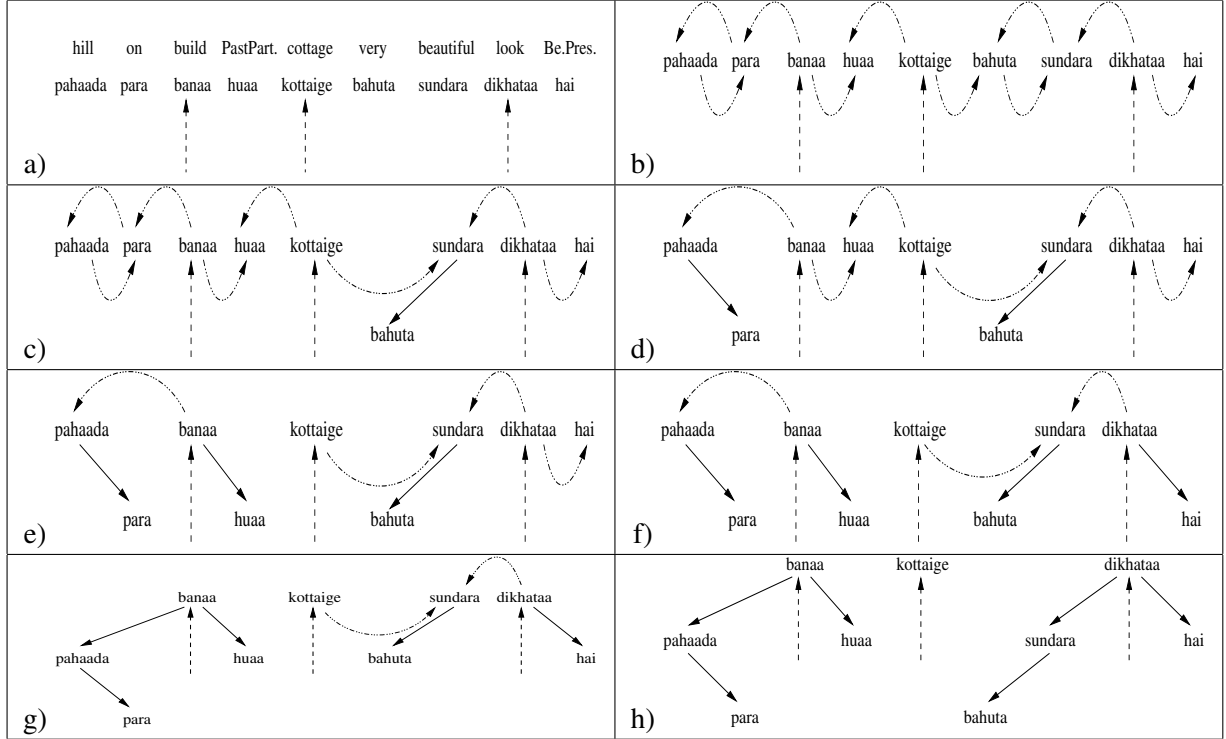


Figure 3: Steps taken by GNPPA. The dashed arcs indicate the unconnected words in *unConn*. The dotted arcs indicate the candidate arcs in *candidateArcs* and the solid arcs are the high scoring arcs that are stored in *builtPPs*

headedness, acyclicity and projectivity constraints while not necessarily being connected i.e. all the words need not have parents.

Given a sentence $W=w_0 \cdots w_n$ with a set of directed arcs A on the words in W , $w_i \rightarrow w_j$ denotes a dependency arc from w_i to w_j , $(w_i, w_j) \in A$. w_i is the parent in the arc and w_j is the child in the arc. $\overset{*}{\rightarrow}$ denotes the reflexive and transitive closure of the arc. $w_i \overset{*}{\rightarrow} w_j$ says that w_i dominates w_j , i.e. there is (possibly empty) path from w_i to w_j .

A node w_i is *unconnected* if it does not have an incoming arc. R is the set of all such unconnected nodes in the dependency graph. For the example in Figure 2, $R=\{\text{banaa}, \text{kottaige}, \text{dikhataa}\}$. A *partial parse* rooted at node w_i denoted by $\rho(w_i)$ is the set of arcs that can be traversed from node w_i . The *yield* of a partial parse $\rho(w_i)$ is the set of nodes dominated by it. We use $\pi(w_i)$ to refer to the yield of $\rho(w_i)$ arranged in the linear order of their occurrence in the sentence. The *span* of the partial tree is the first and last words in its yield.

The dependency graph D can now be represented in terms of partial parses by $D = (W, R, \varrho(R))$ where $W=\{w_0 \cdots w_n\}$ is the sen-

tence, $R=\{r_1 \cdots r_m\}$ is the set of unconnected nodes and $\varrho(R)=\{\rho(r_1) \cdots \rho(r_m)\}$ is the set of partial parses rooted at these unconnected nodes. w_0 is a dummy word added at the beginning of W to behave as a root of a fully connected parse. A fully connected dependency graph would have only one element w_0 in R and the dependency graph rooted at w_0 as the only (fully connected) parse in $\varrho(R)$.

We assume the combined yield of $\varrho(R)$ spans the entire sentence and each of the partial parses in $\varrho(R)$ to be contiguous and non-overlapping with one another. A partial parse is *contiguous* if its yield is contiguous i.e. if a node $w_j \in \pi(w_i)$, then all the words between w_i and w_j also belong to $\pi(w_i)$. A partial parse $\rho(w_i)$ is *non-overlapping* if the intersection of its yield $\pi(w_i)$ with yields of all other partial parses is empty.

3.1 Greedy Non-directional Partial Parsing Algorithm (GNPPA)

Given the sentence W and the set of unconnected nodes R , the parser follows a non-directional greedy approach to establish relations in a bottom up manner. The parser does a greedy search over all the possible relations and picks the one with

the highest score at each stage. This process is repeated until parents for all the nodes that do not belong to R are chosen.

Algorithm 1 lists the outline of the greedy non-directional partial parsing algorithm (GNPPA). *builtPPs* maintains a list of all the partial parses that have been built. It is initialized in line 1 by considering each word as a separate partial parse with just one node. *candidateArcs* stores all the arcs that are possible at each stage of the parsing process in a bottom up strategy. It is initialized in line 2 using the method *initCandidateArcs*($w_0 \dots w_n$). *initCandidateArcs*($w_0 \dots w_n$) adds two candidate arcs for each pair of consecutive words with each other as parent (see Figure 3b). If an arc has one of the nodes in R as the child, it isn't included in *candidateArcs*.

Algorithm 1 Partial Parsing Algorithm

Input: sentence $w_0 \dots w_n$ and set of partial tree roots $unConn = \{r_1 \dots r_m\}$
Output: set of partial parses whose roots are in $unConn$ ($builtPPs = \{\rho(r_1) \dots \rho(r_m)\}$)
1: $builtPPs = \{\rho(w_0) \dots \rho(w_n)\} \leftarrow \{w_0 \dots w_n\}$
2: $candidateArcs = \text{initCandidateArcs}(w_0 \dots w_n)$
3: **while** $candidateArcs.isNotEmpty()$ **do**
4: $bestArc = \underset{c_i \in candidateArcs}{\text{argmax}} \text{score}(c_i, \vec{w})$
5: $builtPPs.remove(bestArc.child)$
6: $builtPPs.remove(bestArc.parent)$
7: $builtPPs.add(bestArc)$
8: $\text{updateCandidateArcs}(bestArc, candidateArcs, builtPPs, unConn)$
9: **end while**
10: **return** $builtPPs$

Once initialized, the candidate arc with the highest score (line 4) is chosen and accepted into *builtPPs*. This involves replacing the best arc's child partial parse $\rho(arc.child)$ and parent partial parse $\rho(arc.parent)$ over which the arc has been formed with the arc $\rho(arc.parent) \rightarrow \rho(arc.child)$ itself in *builtPPs* (lines 5-7). In Figure 3f, to accept the best candidate arc $\rho(banaa) \rightarrow \rho(pahaada)$, the parser would remove the nodes $\rho(banaa)$ and $\rho(pahaada)$ in *builtPPs* and add $\rho(banaa) \rightarrow \rho(pahaada)$ to *builtPPs* (see Figure 3g).

After the best arc is accepted, the *candidateArcs* has to be updated (line 8) to remove the arcs that are no longer valid and add new arcs in the context of the updated *builtPPs*. Algorithm 2 shows the update procedure. First, all the arcs that end on the child are removed (lines 3-7) along with the arc from child to parent. Then, the immedi-

ately previous and next partial parses of the best arc in *builtPPs* are retrieved (lines 8-9) to add possible candidate arcs between them and the partial parse representing the best arc (lines 10-23). In the example, between Figures 3b and 3c, the arcs $\rho(kottaige) \rightarrow \rho(bahuta)$ and $\rho(bahuta) \rightarrow \rho(sundara)$ are first removed and the arc $\rho(kottaige) \rightarrow \rho(sundara)$ is added to *candidateArcs*. Care is taken to avoid adding arcs that end on unconnected nodes listed in R .

The entire GNPPA parsing process for the example sentence in Figure 2 is shown in Figure 3.

Algorithm 2 updateCandidateArcs(*bestArc*, *candidateArcs*, *builtPPs*, *unConn*)

```

1:  $baChild = bestArc.child$ 
2:  $baParent = bestArc.parent$ 
3: for all  $arc \in candidateArcs$  do
4:     if  $arc.child = baChild$  or
         $(arc.parent = baChild$  and
         $arc.child = baParent)$  then
5:          $remove\ arc$ 
6:     end if
7: end for
8:  $prevPP = builtPPs.previousPP(bestArc)$ 
9:  $nextPP = builtPPs.nextPP(bestArc)$ 
10: if  $bestArc.direction == LEFT$  then
11:      $newArc1 = new\ Arc(prevPP, baParent)$ 
12:      $newArc2 = new\ Arc(baParent, prevPP)$ 
13: end if
14: if  $bestArc.direction == RIGHT$  then
15:      $newArc1 = new\ Arc(nextPP, baParent)$ 
16:      $newArc2 = new\ Arc(baParent, nextPP)$ 
17: end if
18: if  $newArc1.parent \notin unConn$  then
19:      $candidateArcs.add(newArc1)$ 
20: end if
21: if  $newArc2.parent \notin unConn$  then
22:      $candidateArcs.add(newArc2)$ 
23: end if
24: return  $candidateArcs$ 

```

3.2 Learning

The algorithm described in the previous section uses a weight vector \vec{w} to compute the best arc from the list of candidate arcs. This weight vector is learned using a simple Perceptron like algorithm similar to the one used in (Shen and Joshi, 2008). Algorithm 3 lists the learning framework for GNPPA.

For a training sample with sentence $w_0 \dots w_n$, projected partial parses $projectedPPs = \{\rho(r_i) \dots \rho(r_m)\}$, unconnected words $unConn$ and weight vector \vec{w} , the *builtPPs* and *candidateArcs* are initiated as in algorithm 1. Then the arc with the highest score is selected. If this arc belongs to the parses in *projectedPPs*, *builtPPs* and *candidateArcs* are updated similar to the operations in

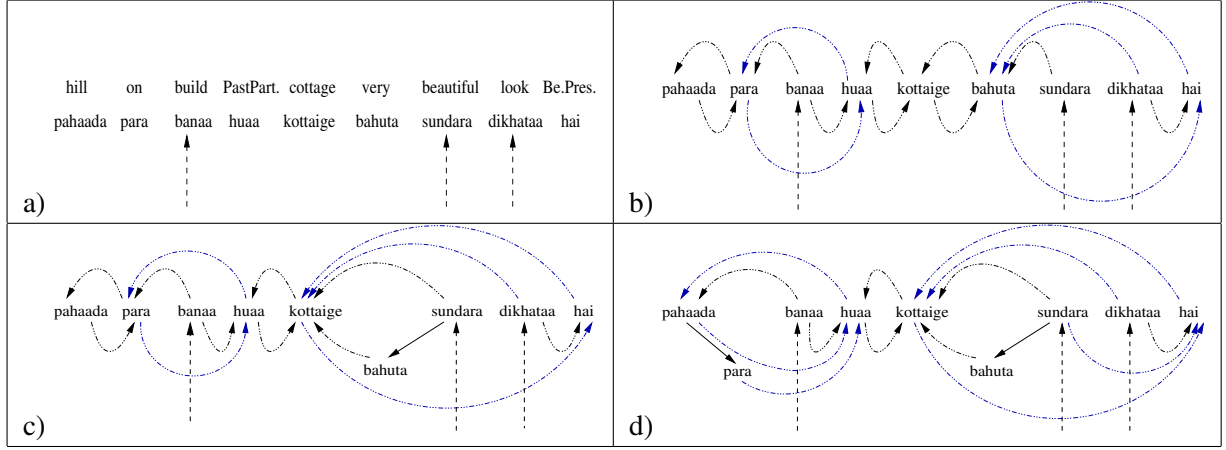


Figure 4: First four steps taken by E-GNPPA. The blue colored dotted arcs are the additional candidate arcs that are added to *candidateArcs*

algorithm 1. If it doesn't, it is treated as a negative sample and a corresponding positive candidate arc which is present both *projectedPPs* and *candidateArcs* is selected (lines 11-12).

The weights of the positive candidate arc are increased while that of the negative sample (best arc) are decreased. To reduce over fitting, we use averaged weights (Collins, 2002) in algorithm 1.

Algorithm 3 Learning for Non-directional Greedy Partial Parsing Algorithm

Input: sentence $w_0 \dots w_n$, projected partial parses *projectedPPs*, unconnected words *unConn*, current \vec{w}

Output: updated \vec{w}

- 1: $builtPPs = \{\rho(r_1) \dots \rho(r_n)\} \leftarrow \{w_0 \dots w_n\}$
 - 2: $candidateArcs = \text{initCandidateArcs}(w_0 \dots w_n)$
 - 3: **while** *candidateArcs.isNotEmpty()* **do**
 - 4: $bestArc = \underset{c_i \in candidateArcs}{\text{argmax}} \text{score}(c_i, \vec{w})$
 - 5: **if** $bestArc \in projectedPPs$ **then**
 - 6: $builtPPs.remove(bestArc.child)$
 - 7: $builtPPs.remove(bestArc.parent)$
 - 8: $builtPPs.add(bestArc)$
 - 9: $\text{updateCandidateArcs}(bestArc, candidateArcs, builtPPs, unConn)$
 - 10: **else**
 - 11: $allowedArcs = \{c_i \mid c_i \in candidateArcs \ \&\& \ c_i \in projectedPPs\}$
 - 12: $compatArc = \underset{c_i \in allowedArcs}{\text{argmax}} \text{score}(c_i, \vec{w})$
 - 13: $\text{promote}(compatArc, \vec{w})$
 - 14: $\text{demote}(bestArc, \vec{w})$
 - 15: **end if**
 - 16: **end while**
 - 17: **return** *builtPPs*
-

3.3 Extended GNPPA (E-GNPPA)

The GNPPA described in section 3.1 assumes that the partial parses are contiguous. The example in Figure 5 has a partial tree $\rho(\text{dikhataa})$ which isn't contiguous. Its yield doesn't con-

tain *bahuta* and *sundara*. We call such *non-contiguous* partial parses whose yields encompass the yield of an other partial parse as *partially contiguous*. Partially contiguous parses are common in the projected data and would not be parsable by the algorithm 1 ($\rho(\text{dikhataa}) \rightarrow \rho(\text{kottaige})$ would not be identified).

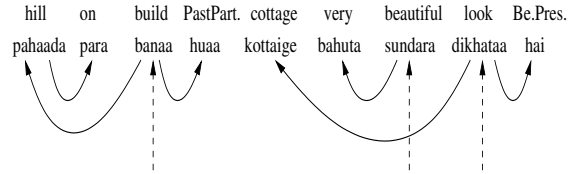


Figure 5: Dependency parse with a partially contiguous partial parse

In order to identify and learn from relations which are part of partially contiguous partial parses, we propose an extension to GNPPA. The extended GNPPA (E-GNPPA) broadens its scope while searching for possible candidate arcs given R and *builtPPs*. If the immediate previous or the next partial parses over which arcs are to be formed are designated unconnected nodes, the parser looks further for a partial parse over which it can form arcs. For example, in Figure 4b, the arc $\rho(\text{para}) \rightarrow \rho(\text{banaa})$ can not be added to the *candidateArcs* since *banaa* is a designated unconnected node in *unConn*. The E-GNPPA looks over the unconnected node and adds the arc $\rho(\text{para}) \rightarrow \rho(\text{huaa})$ to the candidate arcs list *candidateArcs*.

E-GNPPA differs from algorithm 1 in lines 2 and 8. The E-GNPPA uses an extended initialization method $\text{initCandidateArcsExtended}(w_0)$ for

<i>Parent and Child</i>	<i>par.pos, chd.pos, par.lex, chd.lex</i>
<i>Sentence Context</i>	<i>par-1.pos, par-2.pos, par+1.pos, par+2.pos, par-1.lex, par+1.lex chd-1.pos, chd-2.pos, chd+1.pos, chd+2.pos, chd-1.lex, chd+1.lex</i>
<i>Structural Info</i>	<i>leftMostChild(par).pos, rightMostChild(par).pos, leftSibling(chd).pos, rightSibling(chd).pos</i>
<i>Partial Parse Context</i>	<i>previousPP().pos, previousPP().lex, nextPP().pos, nextPP().lex</i>

Table 1: Information on which features are defined. *par* denotes the parent in the relation and *chd* the child. *.pos* and *.lex* is the POS and word-form of the corresponding node. *+/-i* is the previous/next *i*th word in the sentence. *leftMostChild()* and *rightMostChild()* denote the left most and right most children of a node. *leftSibling()* and *rightSibling()* get the immediate left and right siblings of a node. *previousPP()* and *nextPP()* return the immediate previous and next partial parses of the arc in *builtPPs* at the state.

candidateArcs in line 2 and an extended procedure *updateCandidateArcsExtended* to update the *candidateArcs* after each step in line 8. Algorithm 4 shows the changes w.r.t algorithm 2. Figure 4 presents the steps taken by the E-GNPPA parser for the example parse in Figure 5.

Algorithm 4 *updateCandidateArcsExtended*
(*bestArc, candidateArcs, builtPPs, unConn*)

```

... lines 1 to 7 of Algorithm 2 ...
prevPP = builtPPs.previousPP(bestArc)
while prevPP ∈ unConn do
  prevPP = builtPPs.previousPP(prevPP)
end while
nextPP = builtPPs.nextPP(bestArc)
while nextPP ∈ unConn do
  nextPP = builtPPs.nextPP(nextPP)
end while
... lines 10 to 24 of Algorithm 2 ...

```

3.4 Features

Features for a relation (candidate arc) are defined on the POS tags and lexical items of the nodes in the relation and those in its context. Two kinds of context are used a) context from the input sentence (*sentence context*) b) context in *builtPPs* i.e. nearby partial parses (*partial parse context*). Information from the partial parses (*structural info*) such as left and right most children of the parent node in the relation, left and right siblings of the child node in the relation are also used. Table 1 lists the information on which features are defined in the various configurations of the three language parsers. The actual features are combinations of the information present in the table. The set varies depending on the language and whether its GNPPA or E-GNPPA approach.

While training, no features are defined on whether a node is unconnected (present in *un-*

Conn) or not as this information isn't available during testing.

4 Hindi Projected Dependency Treebank

We conducted experiments on English-Hindi parallel data by transferring syntactic information from English to Hindi to build a projected dependency treebank for Hindi.

The TIDES English-Hindi parallel data containing 45,000 sentences was used for this purpose ¹ (Venkatapathy, 2008). Word alignments for these sentences were obtained using the widely used GIZA++ toolkit in grow-diag-final-and mode (Och and Ney, 2003). Since Hindi is a morphologically rich language, root words were used instead of the word forms. A bidirectional English POS tagger (Shen et al., 2007) was used to POS tag the source sentences and the parses were obtained using the first order MST parser (McDonald et al., 2005) trained on dependencies extracted from Penn treebank using the head rules of Yamada and Matsumoto (2003). A CRF based Hindi POS tagger (PVS. and Gali, 2007) was used to POS tag the target sentences.

English and Hindi being morphologically and syntactically divergent makes the word alignment and dependency projection a challenging task. The source dependencies are projected using an approach similar to (Hwa et al., 2005). While they use post-projection transformations on the projected parse to account for annotation differences, we use pre-projection transformations on the source parse. The projection algorithm pro-

¹The original data had 50,000 parallel sentences. It was later refined by IIIT-Hyderabad to remove repetitions and other trivial errors. The corpus is still noisy with typographical errors, mismatched sentences and unfaithful translations.

duces acyclic parses which could be unconnected and non-projective.

4.1 Annotation Differences in Hindi and English

Before projecting the source parses onto the target sentence, the parses are transformed to reflect the annotation scheme differences in English and Hindi. While English dependency parses reflect the PTB annotation style (Marcus et al., 1994), we project them to Hindi to reflect the annotation scheme described in (Begum et al., 2008). The differences in the annotation schemes are with respect to three phenomena: a) head of a verb group containing auxiliary and main verbs, b) prepositions in a prepositional phrase (PP) and c) coordination structures.

In the English parses, the auxiliary verb is the head of the main verb while in Hindi, the main verb is the head of the auxiliary in the verb group. For example, in the Hindi parse in Figure 1, *dikhataa* is the head of the auxiliary verb *hai*. The prepositions in English are realized as post-positions in Hindi. While prepositions are the heads in a preposition phrase, post-positions are the modifiers of the preceding nouns in Hindi. In *pahaada para* (*on the hill*), *hill* is the head of *para*. In coordination structures, while English differentiates between how NP coordination and VP coordination structures behave, Hindi annotation scheme is consistent in its handling. Leftmost verb is the head of a VP coordination structure in English whereas the rightmost noun is the head in case of NP coordination. In Hindi, the conjunct is the head of the two verbs/nouns in the coordination structure.

These three cases are identified in the source tree and appropriate transformations are made to the source parse itself before projecting the relations using word alignments.

5 Experiments

We carried out all our experiments on parallel corpora belonging to English-Hindi, English-Bulgarian and English-Spanish language pairs. While the Hindi projected treebank was obtained using the method described in section 4, Bulgarian and Spanish projected datasets were obtained using the approach in (Ganchev et al., 2009). The datasets of Bulgarian and Spanish that contributed to the best accuracies for Ganchev et al. (2009)

Statistic	Hindi	Bulgarian	Spanish
N(Words)	226852	71986	133124
N(Parents==1)	44607	30268	54815
P(Parents==1)	19.7	42.0	41.1
N(Full trees)	593	1299	327
N(GNPPA)	30063	10850	19622
P(GNPPA)	16.4	26.0	25.0
N(E-GNPPA)	35389	12281	24577
P(E-GNPPA)	19.3	29.4	30.0

Table 2: Statistics of the Hindi, Bulgarian and Spanish projected treebanks used for experiments. Each of them has 10,000 randomly picked parses. $N(X)$ denotes *number of X* and $P(X)$ denotes *percentage of X*. $N(\text{Words})$ is the number of words. $N(\text{Parents}==1)$ is the number of words without a parent. $N(\text{Full trees})$ is the number of parses which are fully connected. $N(\text{GNPPA})$ is the number of relations learnt by GNPPA parser and $N(\text{E-GNPPA})$ is the number of relations learnt by E-GNPPA parser. Note that $P(\text{GNPPA})$ is calculated as $N(\text{GNPPA})/(N(\text{Words}) - N(\text{Parents}==1))$.

were used in our work (7 rules dataset for Bulgarian and 3 rules dataset for Spanish). The Hindi, Bulgarian and Spanish projected dependency treebanks have 44760, 39516 and 76958 sentences respectively. Since we don't have confidence scores for the projections on the sentences, we picked 10,000 sentences randomly in each of the three datasets for training the parsers². Other methods of choosing the 10K sentences such as those with the max. no. of relations, those with least no. of unconnected words, those with max. no. of contiguous partial trees that can be learned by GNPPA parser etc. were tried out. Among all these, random selection was consistent and yielded the best results. The errors introduced in the projected parses by errors in word alignment, source parser and projection are not consistent enough to be exploited to select the better parses from the entire projected data.

Table 2 gives an account of the randomly chosen 10k sentences in terms of the number of words, words without parents etc. Around 40% of the words spread over 88% of sentences in Bulgarian and 97% of sentences in Spanish have no parents. Traditional dependency parsers which only train from fully connected trees would not be able to learn from these sentences. $P(\text{GNPPA})$ is the percentage of relations in the data that are learned by the GNPPA parser satisfying the contiguous partial tree constraint and $P(\text{E-GNPPA})$ is the per-

²Exactly 10K sentences were selected in order to compare our results with those of (Ganchev et al., 2009).

Parser	Hindi		Bulgarian		Spanish	
	<i>Punct</i>	<i>NoPunct</i>	<i>Punct</i>	<i>NoPunct</i>	<i>Punct</i>	<i>NoPunct</i>
Baseline	78.70	77.39	51.85	55.15	41.60	45.61
GNPPA	80.03*	78.81*	77.03*	79.06*	65.49*	68.70*
E-GNPPA	81.10* †	79.94* †	78.93* †	80.11* †	67.69* †	70.90* †

Table 3: UAS for Hindi, Bulgarian and Spanish with the baseline, GNPPA and E-GNPPA parsers trained on 10k parses selected randomly. *Punct* indicates evaluation with punctuation whereas *NoPunct* indicates without punctuation. * next to an accuracy denotes statistically significant (McNemar’s and $p < 0.05$) improvement over the baseline. † denotes significance over GNPPA

centage that satisfies the partially contiguous constraint. E-GNPPA parser learns around 2-5% more no. of relations than GNPPA due to the relaxation in the constraints.

The Hindi test data that was released as part of the ICON-2010 Shared Task (Husain et al., 2010) was used for evaluation. For Bulgarian and Spanish, we used the same test data that was used in the work of Ganchev et al. (2009). These test datasets had sentences from the training section of the CoNLL Shared Task (Nivre et al., 2007) that had lengths less than or equal to 10. All the test datasets have gold POS tags.

A baseline parser was built to compare learning from partial parses with learning from fully connected parses. Full parses are constructed from partial parses in the projected data by randomly assigning parents to unconnected parents, similar to the work in (Hwa et al., 2005). The unconnected words in the parse are selected randomly one by one and are assigned parents randomly to complete the parse. This process is repeated for all the sentences in the three language datasets. The parser is then trained with the GNPPA algorithm on these fully connected parses to be used as the baseline.

Table 3 lists the accuracies of the baseline, GNPPA and E-GNPPA parsers. The accuracies are unlabeled attachment scores (UAS): the percentage of words with the correct head. Table 4 compares our accuracies with those reported in (Ganchev et al., 2009) for Bulgarian and Spanish.

5.1 Discussion

The baseline reported in (Ganchev et al., 2009) significantly outperforms our baseline (see Table 4) due to the different baselines used in both the works. In our work, while creating the data for the baseline by assigning random parents to unconnected words, acyclicity and projectivity con-

Parser	Bulgarian	Spanish
Ganchev-Baseline	72.6	69.0
Baseline	55.15	45.61
Ganchev-Discriminative	78.3	72.3
GNPPA	79.06	68.70
E-GNPPA	80.11	70.90

Table 4: Comparison of baseline, GNPPA and E-GNPPA with baseline and discriminative model from (Ganchev et al., 2009) for Bulgarian and Spanish. Evaluation didn’t include punctuation.

straints are not enforced. Ganchev et al. (2009)’s baseline is similar to the first iteration of their discriminative model and hence performs better than ours. Our Bulgarian E-GNPPA parser achieved a 1.8% gain over theirs while the Spanish results are lower. Though their training data size is also 10K, the training data is different in both our works due to the difference in the method of choosing 10K sentences from the large projected treebanks.

The GNPPA accuracies (see table 3) for all the three languages are significant improvements over the baseline accuracies. This shows that learning from partial parses is effective when compared to imposing the connected constraint on the partially projected dependency parse. Even while projecting source dependencies during data creation, it is better to project high confidence relations than look to project more relations and thereby introduce noise.

The E-GNPPA which also learns from partially contiguous partial parses achieved statistically significant gains for all the three languages. The gains across languages is due to the fact that in the 10K data that was used for training, E-GNPPA parser could learn 2 – 5% more relations over GNPPA (see Table 2).

Figure 6 shows the accuracies of baseline and E-

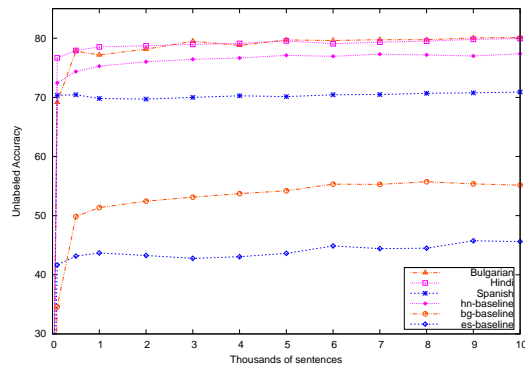


Figure 6: Accuracies (without punctuation) w.r.t varying training data sizes for baseline and E-GNPPA parsers.

GNPPA parser for the three languages when training data size is varied. The parsers peak early with less than 1000 sentences and make small gains with the addition of more data.

6 Conclusion

We presented a non-directional parsing algorithm that can learn from partial parses using syntactic and contextual information as features. A Hindi projected dependency treebank was developed from English-Hindi bilingual data and experiments were conducted for three languages Hindi, Bulgarian and Spanish. Statistically significant improvements were achieved by our partial parsers over the baseline system. The partial parsing algorithms presented in this paper are not specific to bitext projections and can be used for learning from partial parses in any setting.

References

- R. Begum, S. Husain, A. Dhawaj, D. Sharma, L. Bai, and R. Sangal. 2008. Dependency annotation scheme for indian languages. In *Proceedings of The Third International Joint Conference on Natural Language Processing (IJCNLP)*, Hyderabad, India.
- Michael John Collins. 1999. *Head-driven statistical models for natural language parsing*. Ph.D. thesis, University of Pennsylvania, Philadelphia, PA, USA. AAI9926110.
- Michael Collins. 2002. Discriminative training methods for hidden markov models: theory and experiments with perceptron algorithms. In *Proceedings of the ACL-02 conference on Empirical methods in natural language processing - Volume 10*, EMNLP

'02, pages 1–8, Morristown, NJ, USA. Association for Computational Linguistics.

Jason M. Eisner. 1996. Three new probabilistic models for dependency parsing: an exploration. In *Proceedings of the 16th conference on Computational linguistics - Volume 1*, pages 340–345, Morristown, NJ, USA. Association for Computational Linguistics.

Kuzman Ganchev, Jennifer Gillenwater, and Ben Taskar. 2009. Dependency grammar induction via bitext projection constraints. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP: Volume 1 - Volume 1*, ACL-IJCNLP '09, pages 369–377, Morristown, NJ, USA. Association for Computational Linguistics.

Yoav Goldberg and Michael Elhadad. 2010. An efficient algorithm for easy-first non-directional dependency parsing. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, HLT '10, pages 742–750, Morristown, NJ, USA. Association for Computational Linguistics.

Samar Husain, Prashanth Mannem, Bharath Ambati, and Phani Gadde. 2010. Icon 2010 tools contest on indian language dependency parsing. In *Proceedings of ICON 2010 NLP Tools Contest*.

Rebecca Hwa, Philip Resnik, Amy Weinberg, Clara Cabezas, and Okan Kolak. 2005. Bootstrapping parsers via syntactic projection across parallel texts. *Nat. Lang. Eng.*, 11:311–325, September.

Wenbin Jiang and Qun Liu. 2010. Dependency parsing and projection based on word-pair classification. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, ACL '10, pages 12–20, Morristown, NJ, USA. Association for Computational Linguistics.

P. Koehn. 2005. Europarl: A parallel corpus for statistical machine translation. In *MT summit*, volume 5. Citeseer.

Marco Kuhlmann and Joakim Nivre. 2006. Mildly non-projective dependency structures. In *Proceedings of the COLING/ACL on Main conference poster sessions*, pages 507–514, Morristown, NJ, USA. Association for Computational Linguistics.

Mitchell P. Marcus, Beatrice Santorini, and Mary A. Marcinkiewicz. 1994. Building a large annotated corpus of english: The penn treebank. *Computational Linguistics*, 19(2):313–330.

R. McDonald, K. Crammer, and F. Pereira. 2005. Online large-margin training of dependency parsers. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*.

- Jens Nilsson and Joakim Nivre. 2008. Malteval: an evaluation and visualization tool for dependency parsing. In *Proceedings of the Sixth International Language Resources and Evaluation (LREC'08)*, Marrakech, Morocco, may. European Language Resources Association (ELRA). <http://www.lrec-conf.org/proceedings/lrec2008/>.
- Joakim Nivre, Johan Hall, Sandra Kübler, Ryan McDonald, Jens Nilsson, Sebastian Riedel, and Deniz Yuret. 2007. The CoNLL 2007 shared task on dependency parsing. In *Proceedings of the CoNLL Shared Task Session of EMNLP-CoNLL 2007*, pages 915–932, Prague, Czech Republic. Association for Computational Linguistics.
- Joakim Nivre. 2003. An Efficient Algorithm for Projective Dependency Parsing. In *Eighth International Workshop on Parsing Technologies*, Nancy, France.
- Joakim Nivre. 2009. Non-projective dependency parsing in expected linear time. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP*, pages 351–359, Suntec, Singapore, August. Association for Computational Linguistics.
- Franz Josef Och and Hermann Ney. 2003. A systematic comparison of various statistical alignment models. *Computational Linguistics*, 29(1):19–51.
- Avinesh PVS. and Karthik Gali. 2007. Part-Of-Speech Tagging and Chunking using Conditional Random Fields and Transformation-Based Learning. In *Proceedings of the IJCAI and the Workshop On Shallow Parsing for South Asian Languages (SPSAL)*, pages 21–24.
- Roi Reichart and Ari Rappoport. 2007. Self-training for enhancement and domain adaptation of statistical parsers trained on small datasets. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, pages 616–623, Prague, Czech Republic, June. Association for Computational Linguistics.
- Libin Shen and Aravind Joshi. 2008. LTAG dependency parsing with bidirectional incremental construction. In *Proceedings of the 2008 Conference on Empirical Methods in Natural Language Processing*, pages 495–504, Honolulu, Hawaii, October. Association for Computational Linguistics.
- L. Shen, G. Satta, and A. Joshi. 2007. Guided learning for bidirectional sequence classification. In *Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics (ACL)*.
- Mark Steedman, Miles Osborne, Anoop Sarkar, Stephen Clark, Rebecca Hwa, Julia Hockenmaier, Paul Ruhlen, Steven Baker, and Jeremiah Crim. 2003. Bootstrapping statistical parsers from small datasets. In *Proceedings of the tenth conference on European chapter of the Association for Computational Linguistics - Volume 1*, EACL '03, pages 331–338, Morristown, NJ, USA. Association for Computational Linguistics.
- Jrg Tiedemann. 2002. MatsLex - a multilingual lexical database for machine translation. In *Proceedings of the 3rd International Conference on Language Resources and Evaluation (LREC'2002)*, volume VI, pages 1909–1912, Las Palmas de Gran Canaria, Spain, 29-31 May.
- Sriram Venkatapathy. 2008. Nlp tools contest - 2008: Summary. In *Proceedings of ICON 2008 NLP Tools Contest*.
- Hiroyasu Yamada and Yuji Matsumoto. 2003. Statistical Dependency Analysis with Support Vector Machines. In *In Proceedings of IWPT*, pages 195–206.
- David Yarowsky, Grace Ngai, and Richard Wicentowski. 2001. Inducing multilingual text analysis tools via robust projection across aligned corpora. In *Proceedings of the first international conference on Human language technology research, HLT '01*, pages 1–8, Morristown, NJ, USA. Association for Computational Linguistics.