A Graph-based Lattice Dependency Parser for Joint Morphological Segmentation and Syntactic Analysis

Wolfgang Seeker and **Özlem Çetinoğlu**
Institut für Maschinelle Sprachverarbeitung
University of Stuttgart
{seeker,ozlem}@ims.uni-stuttgart.de

Abstract

Space-delimited words in Turkish and Hebrew text can be further segmented into meaningful units, but syntactic and semantic context is necessary to predict segmentation. At the same time, predicting correct syntactic structures relies on correct segmentation. We present a graph-based lattice dependency parser that operates on morphological lattices to represent different segmentations and morphological analyses for a given input sentence. The lattice parser predicts a dependency tree over a path in the lattice and thus solves the joint task of segmentation, morphological analysis, and syntactic parsing. We conduct experiments on the Turkish and the Hebrew treebank and show that the joint model outperforms three state-of-the-art pipeline systems on both data sets. Our work corroborates findings from constituency lattice parsing for Hebrew and presents the first results for full lattice parsing on Turkish.

1 Introduction

Linguistic theory has provided examples from many different languages in which grammatical information is expressed via case marking, morphological agreement, or clitics. In these languages, configurational information is less important than in English since the words are overtly marked for their syntactic relations to each other. Such morphologically rich languages pose many new challenges to today's natural language processing technology, which has often been developed for English.

One of the first challenges is the question on how to represent morphologically rich languages and what are the basic units of analysis (Tsarfaty et al., 2010). The Turkish treebank (Oflazer et al., 2003), for example, represents words as sequences of *inflectional groups*, semantically coherent groups of morphemes separated by derivational boundaries. The treebank for Modern Hebrew (Sima'an et al., 2001) chooses morphemes as the basic unit of representation. A space-delimited word in the treebank can consist of several morphemes that may belong to independent syntactic contexts.

Both Turkish and Hebrew show high amounts of ambiguity when it comes to the correct segmentation of words into inflectional groups and morphemes, respectively. Within a sentence, however, these ambiguities can often be resolved by the syntactic and semantic context in which these words appear.

A standard (dependency) parsing system decides segmentation, morphological analysis (including POS), and syntax one after the other in a pipeline setup. While pipelines are fast and efficient, they cannot model interaction between these different levels of analysis, however. It has therefore been argued that joint modeling of these three tasks is more suitable to the problem (Tsarfaty, 2006). In previous research, several transition-based parsers have been proposed to model POS/morphological tagging and parsing jointly (Hatori et al., 2011; Bohnet and Nivre, 2012; Bohnet et al., 2013). Such parsing systems have been further extended to also solve the segmentation problem in Chinese (Hatori et al., 2012; Li and Zhou, 2012; Zhang et al., 2014). Transition-based parsers are attractive since

Transactions of the Association for Computational Linguistics, vol. 3, pp. 359–373, 2015. Action Editor: Joakim Nivre.
Submission batch: 4/2015; Published 6/2015.

they do not rely on global optimization and thus deal well with the increased model complexity that comes with joint modeling. Nonetheless, graph-based models have been proposed as well, e.g. by Li et al. (2011) for joint POS tagging and dependency parsing. Their parsers model the joint problem directly at the cost of increased model complexity.

In this paper, we present a graph-based dependency parser for lattice parsing that handles the increased complexity by applying dual decomposition. The parser operates on morphological lattices and predicts word segmentation, morphological analysis, and dependency syntax jointly. It decomposes the problem into several subproblems and uses dual decomposition to find a common solution (Koo et al., 2010; Martins et al., 2010). The subproblems are defined such that they can be solved efficiently and agreement is found in an iterative fashion. Decomposing the problem thus keeps the complexity of the joint parser on a tractable level.

We test the parser on the Turkish and the Hebrew treebank. The segmentation problem in these languages can be tackled with the same approach even though their underlying linguistic motivation is quite different. In our experiments, the lattice dependency parser outperforms three state-of-the-art pipeline systems. Lattice parsing for Hebrew has been thoroughly investigated in constituency parsing (Cohen and Smith, 2007; Goldberg and Tsarfaty, 2008; Goldberg and Elhadad, 2013), demonstrating the viability of joint modeling. To the best of our knowledge, our work is the first to apply full lattice parsing to the Turkish treebank.

We introduce the segmentation problem in Turkish and Hebrew in Section 2 and present the lattice parser in Section 3. Sections 4 and 5 describe the experiments and their results and we discuss related work in Section 6. We conclude with Section 7.

2 Word Segmentation in Turkish and Hebrew

A lot of morphosyntactic information is overtly marked on words in morphologically rich languages. It is also common to express syntactic information through derivation or composition. As a consequence these words, orthographically written together, actually have word-internal syntactic structures. Moreover, word-external relations may depend on the word-internal structures, e.g., a word could be grammatically related to only parts of another word instead of the whole.

For instance, in the Turkish sentence *ekmek aldım*, each word has two analyses. *ekmek* means ‘bread’ or the nominal ‘planting’ which is derived from the verb stem *ek* ‘plant’ with the nominalization suffix *mek*. *aldım* has the meaning ‘I bought’ which decomposes as *al-dı-m* ‘buy-Past-1sg’. It also means ‘I was red’, which is derived from the adjective *al* ‘red’, inflected for past tense, 1st person singular.

Depending on the selected morphological analysis for each word, syntax and semantics of the sentence change. When the first analysis is selected for both words, the syntactic representation of the sentence is given in Figure 1, which corresponds to the meaning ‘I bought bread’. When the nominal ‘planting’ is selected for the first word, it is a grammatical sentence albeit with an implausable meaning. When the derivational analysis of the second word is selected, regardless of the morphological analysis of the first word, the sentence is ungrammatical due to subject-verb agreement failure. Although all morphological analyses for these two words are correct in isolation, when they occur in the same syntactic context only some combinations are grammatical.

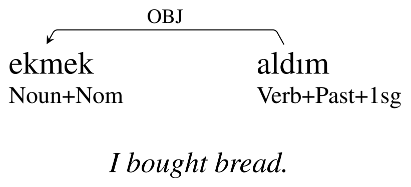

Figure 1: Dependency representation for *ekmek aldım*.

This small example demonstrates that the syntactic structure depends on the morphological disambiguation of the words. At the same time, it shows that syntax can help pick the right morphological analysis.

For a joint system to decide the morphological and syntactic representation together, all possible analyses must be available to the system. The possible morphological analyses of a word can be efficiently represented in a lattice structure. The lattice representation of the sentence in Figure 1 is given in Figure 2, with double circles denoting word boundaries. A sentence lattice is the concatenation of its word lattices. A morphological analysis of a word is a full path from the initial state to the final state of its lattice. Labels on the transitions are the surface form and underlying morphological representation of segments.[1]

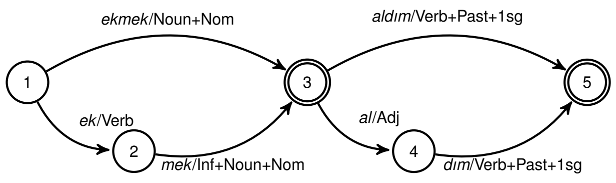

Figure 2: A morphological lattice for *ekmek aldım*.

Lattices also capture well the segmentation of words in Hebrew. Different from Turkish, Hebrew segments can be syntactic units like determiners, prepositions, or relativizers attached to stem segments. In an example given by Goldberg and Tsarfaty (2008), the word *hneim* 'the pleasant/made-pleasant' has three analyses corresponding to the lattice in Figure 3.

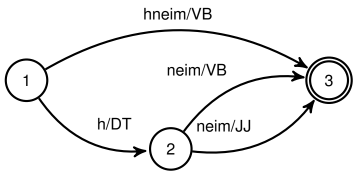

Figure 3: The lattice for *hneim* (Goldberg and Tsarfaty, 2008).

Both the Hebrew and the Turkish treebank annotate dependencies between units smaller than words. In the Turkish treebank, a space-delimited word is segmented into one or more segments depending on its morphological representation. The number of segments is determined by the number of derivations. If it was derived *n* times, it is represented as *n+1* segments. The derivational boundaries are part of the morphological representation. In the Turkish dependency parsing literature (Eryiğit et al., 2008; Çetinoğlu and Kuhn, 2013) these segments are called *inflectional groups* (IGs). IGs consist of one or more inflectional morphemes. The head of a non-final IG is the IG to its right with a dependency relation DERIV. The head of a final IG could be any IG of another word.

The Hebrew treebank defines relations between *morphemes* (Sima'an et al., 2001). Those morphemes correspond to what is usually considered a separate syntactic unit in English. In Hebrew script, word classes like prepositions and conjunctions are always written together with the following word. Contrary to Turkish, syntactic heads of both non-final and final segments can be internal or external to the same space-delimited word.

For convenience, we will use *token* to refer to the smallest unit of processing for the remainder of the paper. It corresponds to IGs in Turkish and morphemes in Hebrew. A transition in a morphological lattice therefore represents one token. We will use *word* to refer to space-delimited words.[2] In standard parsing, these two terms usually coincide with a token in a sentence being separated from the surrounding ones by space.

3 Lattice Parsing

One can think of lattice parsing as two tasks that the parser solves simultaneously: the parser needs to *find a path* through the lattice and it needs to *find a parse tree*. Importantly, the parser solves this task under the condition that the parse tree and the path agree with each other, i.e. the tokens that the parse tree spans over must form the path through the lattice. Decomposing the problem in this way defines the three components for the parser.

Let x be an input lattice and $T = \{\textsc{root}, t_1, t_2, \ldots, t_n\}$ be the set of tokens in x. In what follows, we assume two different structures, lattices and dependency trees. Dependency trees are represented as directed acyclic trees with a special root node (ROOT), whereas lattices are directed acyclic graphs with one defined start state and one defined end state (see Figures 2 and 3). For dependency trees, we will use the terms *node* and *arc* to refer to the vertices and the edges between the vertices, respectively. Tokens are represented as

[1]Surface forms on the transitions are given for convenience. In the Turkish treebank, only final segments have surface forms (of full words), the surface forms of non-final segments are represented as underscores.

[2]This is a technical definition of word and has no ambition to make claims about the linguistic definition of a word.

nodes in the dependency tree. For lattices, we use the terms *state* and *transition* to refer to the vertices and their edges in the lattice. Tokens are represented as transitions between states in the lattice.

Find The Path. A token bigram in a lattice x is a pair of two transitions $\langle t, t' \rangle$, such that the target state of t in x coincides with the source state of t' in x. A chain of overlapping bigrams that starts from the initial state and ends in the final state forms a path through the lattice. We represent the ROOT token as the first transition, i.e. a single transition that leaves the initial state of the lattice.

Given a lattice x, we define the index set of token bigrams in the lattice to be $S := \{ \langle t, t' \rangle \mid t, t' \in T, \mathrm{target}(x,t) = \mathrm{source}(x,t') \}$. For later, we furthermore define $S_{|t} := \{ \langle k, t \rangle \mid \langle k, t \rangle \in S,\ k \in T \}$ to be the set of bigrams that have t at the second position. A consecutive path through the lattice is defined as an indicator vector $\boldsymbol{p} := \langle p_s \rangle_{s \in S}$ where $p_s = 1$ means that bigram s is part of the path, otherwise $p_s = 0$. We define $\mathcal{P}$ as the set of all well-formed paths, i.e. all paths that lead from the initial to the final state.

We use a linear model that factors over bigrams. Given a scoring function f_{P} that assigns scores to paths, the path with the highest score can be found by

$$\hat{\boldsymbol{p}} = \arg\max_{\boldsymbol{p} \in \mathcal{P}} f_{\mathrm{P}}(\boldsymbol{p})$$
$$\text{with} \quad f_{\mathrm{P}}(\boldsymbol{p}) = \sum_{s \in S} p_s\, \boldsymbol{w} \cdot \phi_{\mathrm{SEG}}(s)$$

where ϕ_{SEG} is the feature extraction function for token bigrams. The highest-scoring path through the lattice can be found with the Viterbi algorithm. We use this bigram model later also as a standalone disambiguator for morphological lattices to find the highest-scoring path in a lattice.

Find The Tree. We define the index set of arcs in a dependency tree as $A := \{ \langle h, d, l \rangle \mid h \in T, d \in T - \{\mathrm{ROOT}\}, l \in L, h \neq d \}$ with L being a set of dependency relations. A dependency tree is defined as an indicator vector $\boldsymbol{y} := \langle y_a \rangle_{a \in A}$ where $y_a = 1$ means that arc a is in the parse, otherwise $y_a = 0$. We define $\mathcal{Y}$ to be the set of all well-formed dependency trees.

We follow Koo et al. (2010) and assume an arc-factored model (McDonald et al., 2005) to find the highest-scoring parse. Given a scoring function f_{T} that assigns scores to parses, the problem of finding the highest scoring parse is defined as

$$\hat{\boldsymbol{y}} = \arg\max_{\boldsymbol{y} \in \mathcal{Y}} f_{\mathrm{T}}(\boldsymbol{y})$$
$$\text{with} \quad f_{\mathrm{T}}(\boldsymbol{y}) = \sum_{a \in A} y_a\, \boldsymbol{w} \cdot \phi_{\mathrm{ARC}}(a)$$

where ϕ_{ARC} is the feature extraction function for single arcs and w is the weight vector. We use the Chu-Liu-Edmonds algorithm (CLE) to find the highest-scoring parse (Chu and Liu, 1965; Edmonds, 1967). Note that the algorithm includes all tokens of the lattice into the spanning tree, not just some tokens on some path. Chu-Liu-Edmonds furthermore enforces the tree properties of the output, i.e. acyclicity and exactly one head per token.

Agreement Constraints. To make the path and the parse tree agree with each other, we introduce an additional dependency relation NOREL into L. We define a token that is attached to ROOT with relation NOREL to be *not on the path through the lattice*. These arcs are not scored by the statistical model, they simply serve as a means for CLE to mark tokens as not being part of the path by attaching them to ROOT with this relation. The parser can predict the NOREL label only on arcs attached to root.

We introduce two agreement constraints to ensure that (i) all tokens not on the path are marked with NOREL and must be attached to ROOT and (ii) tokens cannot be dependents of tokens marked with NOREL.

The first constraint is implemented as an *XOR* ($\oplus$) *factor* (Martins et al., 2011b) over token bigrams and arcs. It states that for a token t, either one of its bigrams[3] or its NOREL-arc must be active. There is one such constraint for each token in the lattice.

$$\bigoplus_{s \in S_{|t}} p_s \oplus y_{\langle \mathrm{ROOT}, t, \mathrm{NOREL} \rangle} \quad \text{for all } t \in T \qquad (1)$$

The second constraint ensures that a token that is part of the path will not be attached to a token that

[3]The lattice ensures that always only one of the bigrams with the same token in second position can be part of a path.

is not. It thus guarantees the coherence of the dependency tree over the path through the lattice. It is implemented as an *implication* ($\Longrightarrow$) *factor* (Martins et al., 2015). It states that an active NOREL arc for a token h implies an inactive arc for all arcs having h as head. There is one such constraint for each possible arc in the parse.

$$y_{\langle \text{ROOT}, h, \text{NOREL} \rangle} \implies \neg y_{\langle h,d,l \rangle} \quad (2)$$
$$\text{for all} \quad \langle h,d,l \rangle \in A, h \neq \text{ROOT}, l \neq \text{NOREL}$$

Deciding on a path through the lattice partitions the tokens into two groups: the ones on the path and the ones that are not. By means of the NOREL label, the CLE is also able to partition the tokens into two groups: the ROOT-NOREL tokens and the proper dependency tree tokens. The two agreement constraints then make sure that the two partionings agree with each other. The first constraint explicitly links the two partitionings by requiring each token to either belong to the path or to the ROOT-NOREL tokens. The second constraint ensures that the partitioning by the CLE is consistent, i.e. tokens attached to ROOT with NOREL cannot mix with the other tokens in the tree structure. Before the parser outputs the parse the tokens that do not belong to the path/tree are discarded.

The objective function of the lattice parser is

$$\underset{\boldsymbol{y} \in \mathcal{Y}, \boldsymbol{p} \in \mathcal{P}}{\arg\max} f_{\text{T}}(\boldsymbol{y}) + f_{\text{P}}(\boldsymbol{p})$$

subject to the two agreement constraints in Equations (1) and (2).

We use *Alternating Directions Dual Decomposition* or *AD*^{3} (Martins et al., 2011a)[4] to find the optimal solution to this constrained optimization problem. CLE can be implemented such that its worst case complexity is $\mathcal{O}(T^2)$, while the Viterbi algorithm needed to find the path is of worst case complexity $\mathcal{O}(QT^2)$, where Q is the number of states in the lattice. Instead of combining these two problems directly, which would multiply their complexity, AD^3 combines them additively, such that the complexity of the parser is $\mathcal{O}(k(T^2 + QT^2))$ with k being the number of iterations that AD^3 is run.

[4]http://www.ark.cs.cmu.edu/AD3/

Second-order Parsing. To facilitate second-order features, we use grandparent-sibling head automata as proposed in Koo et al. (2010), which we extend to include dependency relations. The head automata allow the parser to model consecutive sibling and grandparent relations. The architecture of the parser does not need to be changed at all to include the second-order factors. The head automata are simply another component. They compute solutions over the same set of arc indicator variables as the CLE and AD^3 thus ensures that the output of the two algorithms agrees on the tree structure (Koo et al., 2010). The second-order factors dominate the complexity of the entire parser, since solving the head automata is of complexity $\mathcal{O}(T^4L)$.

Pruning. We use rule-based and heuristics-based pruning to reduce the search space of the parser. Arcs between tokens that lie on competing paths through the lattice are cut away as these tokens can never be in a syntactic relation. For the Turkish treebank, we introduce an additional rule based on the annotation scheme of the treebank. In the treebank, the IGs of a word form a chain with each IG having their head immediately to the right and only the last IG choosing the head freely. For the non-final IGs, we therefore restrict the head choice to all IGs that can immediately follow it in the lattice.

In order to restrict the number of heads, we train a simple pairwise classifier that predicts the 10 best heads for each token. It uses the first-order features of the parser's feature model.

Feature Model. The parser extracts features for bigrams (path), arcs (first-order), consecutive siblings, and grandparent relations (both second order). It uses standard features like word form, lemma, POS, morphological features, head direction, and combinations thereof.

Context features are more difficult in lattice parsing than in standard parsing as the left and right context of a token is not specified before parsing. We first extracted context features from all tokens that can follow or precede a token in the lattice. This led to overfitting effects as the model was learning specific lattice patterns. We therefore use latent left and right context and extract features from only one of the left/right neighbor tokens. The latent context is the left/right context token with the highest score from the path features (raw bigram scores, they are not changed by AD^3). The parser extracts context from one token in each direction.

Distance features are also more difficult in lattices since the linear distance between two tokens depends on the actual path chosen by the parser. We define distance simply as the length of the shortest path between two tokens in the lattice, but this distance may not coincide with the actual path.

Context features and distance features show that lattice dependency parsing poses interesting new challenges to feature design. Using latent context features is one way of handling uncertain context, compare also the *delayed features* in Hatori et al. (2011). A thorough investigation of different options is needed here.

Learning. We train a discriminative linear model using passive-aggressive online learning (Crammer et al., 2003) with cost-augmented inference (Taskar et al., 2005) and parameter averaging (Freund and Schapire, 1999). We use Hamming loss over the arcs of the parse tree excluding NOREL arcs. The model trains one parameter vector that includes features from the tree and from the path.

The maximum number of iterations of AD^3 is set to 1000 during training and testing. The algorithm sometimes outputs fractional solutions. During training, the model is updated with these fractional solutions, weighting the features and the loss accordingly. During testing, fractional solutions are projected to an integer solution by first running the best-path algorithm with the path posteriors output by AD^3 and afterwards running CLE on the selected path weighted by the arc posteriors (Martins et al., 2009). In the experiments, fractional solutions occur in about 9% of the sentences in the Turkish development set during testing.

4 Experimental Setup

4.1 The Turkish Data

The training set for Turkish is the 5,635 sentences of the METU-Sabancı Turkish Treebank (Oflazer et al., 2003). The 300 sentences of the ITU validation set (Eryiğit, 2012) are used for testing. As there is no separate development set, we split the training set into 10 parts and used 2 of them as development data. All models run on this development set are trained on the remaining 8 parts. We also report results from 10-fold crossvalidation on the full training set (10cv).

We use the *detached* version of the Turkish treebank (Eryiğit et al., 2011) where multiword expressions are represented as separate tokens. The training set of this version contains 49 sentences with loops. We manually corrected these sentences and use the corrected version in our experiments.[5]

The Turkish raw input is first passed through a morphological analyzer (Oflazer, 1994) in order to create morphological lattices as input to the parser. Gold analyses are added to the training lattices if the morphological analyzer failed to output the correct analyses.

For the pipeline systems, the input lattices are disambiguated by running a morphological disambiguator. We train our own disambiguator using the bigram model from the parser and find the best path through the lattice with the Viterbi algorithm. The disambiguator uses the same bigram features as the lattice parser. The morphological disambiguator is trained on the Turkish treebank as in Çetinoğlu (2014).

4.2 The Hebrew Data

The data for Hebrew comes from the SPMRL Shared Task 2014 (Seddah et al., 2014), which is based on the treebank for Modern Hebrew (Sima'an et al., 2001). It provides lattices and predisambiguated input files. The training and development lattices contained a number of circular structures due to self-loops in some states. We automatically removed the transitions causing these cycles.

Input lattices for training were prepared as for Turkish by adding the gold standard paths if necessary. Compared to the Turkish data, the Hebrew lattices are so large that training times for the lattice parser became unacceptable. We therefore used our morphological disambiguator to predict the 10 best paths for each lattice. All transitions in the lattice that were not part of one of these 10 paths were discarded. Note that the number of actual paths in these pruned lattices is much higher than 10, since the paths converge after each word. All experiments

[5]The corrected version is available on the second author's webpage.

with the joint model for Hebrew are conducted on the pruned lattices. As for Turkish we preprocess the input lattices for all baselines with our own morphological disambiguator.

4.3 Baselines

We compare the lattice parser (JOINT for Turkish, JOINT10 for Hebrew) to three baselines: MATE, TURBO, and PIPELINE.

The first two baseline systems are off-the-shelf dependency parsers that currently represent the state-of-the-art. Mate parser[6] (Bohnet, 2009; Bohnet, 2010) is a graph-based dependency parser that uses Carreras' decoder (Carreras, 2007) and approximate search (McDonald and Pereira, 2006) to produce non-projective dependency structures. TurboParser[7] (Martins et al., 2013) is a graph-based parser that uses a dual decomposition approach and outputs non-projective structures natively. The third baseline system runs the lattice parser on a pre-disambiguated lattice, i.e. in a pipeline setup.

All three baselines are pipeline setups and use the same disambiguator to predict a path through the lattice. The bigram features in the disambiguator are the same as in the joint model. There is thus no difference between the lattice parser and the baselines with respect to the features that are available during segmentation. As opposed to lattice parsing, baseline systems are trained on the gold standard segmentation (and thus gold morphological analyses) in the training data, since automatically predicted paths would not guarantee to be compatible with the gold dependency structures.

The purpose of the first two baselines is to compare the joint parser to the current state-of-the-art. However, the feature sets are different between the joint parser and the off-the-shelf baselines. A difference in performance between the joint parser and the first two baseline systems may thus simply be caused by a difference in the feature set. The third baseline eliminates this difference in the feature sets since it is the actual lattice parser that is run on a disambiguated lattice. Because the morphological disambiguator for the PIPELINE baseline is using the same feature set as the lattice parser (the bigram model), the fact that the joint parser is trained and tested on full lattices is the only difference between these two systems. The PIPELINE baseline thus allows us to test directly the effect of joint decoding compared to a pipeline setup.

[6]`http://code.google.com/p/mate-tools`

[7]`http://www.ark.cs.cmu.edu/TurboParser/`, version 2.0.1

4.4 Evaluation

Standard labeled and unlabeled attachment scores are not applicable when parsing with uncertain segmentation since the number of tokens in the output of the parser may not coincide with the number of tokens in the gold standard. Previous work therefore suggests alternative methods for evaluation, e.g. by means of precision, recall, and f-score over tokens, see e.g. Tsarfaty (2006) or Cohen and Smith (2007).

The uncertainty of segmentation furthermore makes it very hard to evaluate the other levels of analysis independently of the segmentation. In order to decide whether the morphological analysis of a token (or its syntactic attachment) is correct, one always needs to find out first to which token in the gold standard it corresponds. By establishing this correspondence, the segmentation is already being evaluated. Evaluating syntax isolated from the other levels of analysis is therefore not possible in general.

Hatori et al. (2012) count a dependency relation correct only when both the head and the dependent have the correct morphological analysis (here POS) and segmentation. Goldberg (2011, page 53) proposes a similar approach, but only requires surface forms to match between gold standard and prediction. These metrics compute precision and recall over tokens. Eryiğit et al. (2008) and Eryiğit (2012) define an accuracy (IGeval) for Turkish parsing by taking advantage of the annotation scheme in the Turkish treebank: A non-final IG in the Turkish treebank always has its head immediately to the right, always with the same label, which makes it possible to ignore the inner dependency relations, i.e. the segmentation, of a dependent word. The metric therefore only needs to check for each word whether the head of the last IG is attached to the correct IG in another word. The metric includes a back-off strategy in case the head word's segmentation is wrong. A dependency arc is then counted as correct if it attaches to an IG in the correct word and the POS tag of the head IG is the same as in the gold standard.

Parsing Evaluation. We follow Hatori et al. (2012) and use a strict definition of precision and recall (PREC, REC, F1) over tokens to evaluate the full task. We first align the tokens of a word in the parser output with the tokens of the corresponding word in the gold standard using the Needleman-Wunsch algorithm (Needleman and Wunsch, 1970), which we modify so it does not allow for mismatches. A token in the parser output that is not in the gold standard is thus paired with a gap and vice versa. Two tokens must have the same morphological analysis in order to match.[8]

A true positive is defined as a pair of matching tokens whose heads are also aligned and match. For labeled scores, the dependency relations must match as well. Precision is defined as the number of true positives over the number of tokens in the prediction, recall is defined as the number of true positives over the number of tokens in the gold standard. F-score is the harmonic mean of precision and recall.

This metric is very strict and requires all levels of analysis to be correct. In order to evaluate the syntax as independently as possible, we furthermore report IGeval for Turkish, with and without the aforementioned backoff strategy (IGeval and IGeval STRICT). For Hebrew, we report on a version of precision and recall as defined above that only requires the surface forms of the tokens to match.[9] This metric is almost the one proposed in Goldberg (2011). All reported evaluation metrics ignore punctuation.

We do not use TedEval as defined in Tsarfaty et al. (2012) even though it has been used previously to evaluate dependency parsing with uncertain segmentation (Seddah et al., 2013; Zhang et al., 2015). The reason is that it is not an inherently dependency-based framework and the conversion from constituency structures to dependency structures interferes with the metric.[10] The metric proposed in Goldberg (2011) implements the same ideas without edit distance and is defined directly for dependencies.

Segmentation Evaluation. We use the same token-based precision and recall to measure the quality of segmentation and morphological analysis without syntax. For a token to be correct, it has to have the same morphological analysis as the token in the gold standard to which it is aligned. We furthermore report word accuracy (ACC_w), which is the percentage of words that received the correct segmentation.

5 Results

Segmentation and Morphology. Table 1 shows the quality of segmentation and morphological analysis. The baseline for Turkish is the Turkish morphological disambiguator by Sak et al. (2008), trained on the Turkish treebank. For Hebrew, the baseline is the disambiguated lattices provided by the SPMRL 2014 Shared Task.[11] The bigram model is our own morphological disambiguator. The joint model is the full lattice parser, which has access to syntactic information.

The results show that the bigram model is clearly outperforming the baselines for both languages. The feature model of the bigram model was developed on the Turkish development set, but the model also works well for Hebrew. Comparing the bigram model to the joint model shows that overall, the joint model performs better than the bigram model. However, the joint model mainly scores in recall rather than in precision, the bigram model is even ahead of the joint model in precision for Hebrew. The joint model outperforms the bigram model and the baseline also in word accuracy. The results demonstrate that syntactic information is relevant to resolve ambiguity in segmentation and morphology for Turkish and Hebrew.

[8]The method does not create cross, many-to-one, or one-to-many alignments, which can be important because in very rare cases the same token occurs twice in one word.

[9]The metric would not work for Turkish, as the surface forms of non-final IGs are all represented as underscores.

[10]As an experiment, we took a Turkish treebank tree and created artificial parses by attaching one token to a different head each time. All other tokens remained attached to their correct head, and segmentation is kept gold. This gave us 11 parses that contained exactly one attachment error and one parse identical with the gold standard. Running TedEval on each of the 11 incorrect parses gave us 5 different scores. The differences are caused by the transformation of dependency trees to constituency trees, because the constituency trees have different edit distances compared to the gold standard. Consequently, this means that some attachment errors of the dependency parser are punished more than others in an unpredictable way.

[11]A description on how these lattice are produced is given in Seddah et al. (2013, page 159)

		Turkish				Hebrew			
data	system	PREC	REC	F1	ACC$_w$	PREC	REC	F1	ACC$_w$
dev	BASELINE	89.59	88.14	88.86	87.97	85.99	84.07	85.02	80.30
	BIGRAM MODEL	90.69	89.52	90.10	89.45	**86.84**	86.30	86.57	83.46
	JOINT MODEL	**90.80**	**90.22**	**90.51**	**89.94**	86.68	**87.49**	**87.08**	**84.67**
test	BASELINE	89.46	88.51	88.99	87.95	81.79	79.83	80.80	74.85
	BIGRAM MODEL	89.96	89.23	89.59	88.71	**84.44**	83.22	83.83	79.60
	JOINT MODEL	**90.19**	**89.74**	**89.97**	**89.25**	83.88	**83.99**	**83.94**	**80.28**

Table 1: Path selection quality.

		LABELED			UNLABELED			IGeval STRICT		IGeval	
data	system	PREC	REC	F1	PREC	REC	F1	UAS$_{IG}$	LAS$_{IG}$	UAS$_{IG}$	LAS$_{IG}$
dev	MATE	62.54	61.73	62.14	69.44	68.54	68.98	70.60	60.10	74.88	63.46
	TURBO	63.54	62.71	63.12	70.68	69.76	70.22	72.22	61.24	76.58	64.73
	PIPELINE	63.86	63.03	63.44	70.65	69.73	70.19	72.26	61.82	**76.64**	65.49
	JOINT	**64.21**	**63.79***	**64.00**	**70.96**	**70.50***	**70.73**	**72.66***	**62.40**	76.61	**65.59**
10cv	MATE	63.28	62.49	62.88	70.37	69.50	69.94	71.75	61.26	75.84	64.42
	TURBO	63.82	63.03	63.42	71.12	70.24	70.68	72.77	61.89	76.93	65.09
	PIPELINE	64.97	64.17	64.57	71.71	70.83	71.27	73.66	63.52	77.68	66.82
	JOINT	**65.27**	**64.84**†	**65.06**	**72.05***	**71.58**†	**71.82**	**73.93**	**63.85**	**77.74**	**66.83**
test	MATE	64.64	64.12	64.38	70.62	70.04	70.33	71.99	61.84	77.08	65.98
	TURBO	65.36	64.83	65.09	71.66	71.08	71.37	73.16	62.76	78.37	67.02
	PIPELINE	66.40	65.86	66.13	72.30	71.72	72.01	74.33	64.40	**79.61**	**69.02**
	JOINT	**67.33**	**66.99***	**67.16**	**72.94***	**72.58***	**72.76**	**75.02**	**65.32**	79.45	68.99

Table 2: Parsing results for Turkish. Statistically significant differences between the joint system and the pipeline system are marked with † ($p < 0.01$) and $*$ ($p < 0.05$). Significance testing was performed using the Wilcoxon Signed Rank Test (not for F1).

Turkish. Table 2 presents the results of the evaluation of the three baseline systems and the lattice parser on the Turkish data. The PIPELINE and the JOINT system give better results than the other two baselines across the board. This shows that the feature set of the lattice parser is better suited to the Turkish treebank than the feature set of Mate parser and Turbo parser. It is not a surprising result though, since the lattice parser was developed for Turkish whereas the other two parsers were developed for other treebanks.

The JOINT system outperforms the PIPELINE system with respect to the first three metrics. These metrics evaluate syntax, segmentation, and morphological analysis jointly. Higher scores here mean that these aspects in combination have become better. The differences between the PIPELINE and the JOINT model are consistently statistically significant with respect to recall, but only in some cases with respect to precision. The syntactic information that is available to the joint model thus seems to improve recall rather than precision.

The last two columns in Table 2 show an evaluation using IGeval. The IGeval metric is designed to evaluate the syntactic quality with less attention to morphological analysis and segmentation. Here, both PIPELINE and JOINT achieve very similar results and none of the differences is statistical significant. These results suggest that a good part of the improvements in the lattice parser occurs in the morphological analysis/segmentation, whereas the quality of syntactic annotation basically stays the same between the pipeline and the joint model.

Hebrew. The experimental results on the Hebrew data are shown in Table 3. The three baselines perform very similarly. All three baseline systems are run on the output of the same disambiguator, which means that the feature models of the parsers seem to be equally well suited to the Hebrew treebank. The feature model of the lattice parser that is used in the PIPELINE baseline was not adapted to Hebrew in any way, but was used as it was developed for the Turkish data.

Compared to the three baselines, the joint model outperforms them for both labeled and unlabeled scores. As the only difference between PIPELINE and JOINT is the fact that the latter performs joint decoding, the results support the findings in constituency parsing by Tsarfaty (2006), Cohen and Smith (2007), and Goldberg and Tsarfaty (2008), namely that joint decoding is a better model for Hebrew parsing. Judging from statistical significance, the JOINT model improves recall rather than precision, a picture that we found for Turkish as well.

		LABELED			UNLABELED		
data	system	PREC	REC	F1	PREC	REC	F1
dev	MATE	65.41	65.00	65.20	70.65	70.21	70.43
	TURBO	65.12	64.72	64.92	70.44	70.00	70.22
	PIPELINE	65.64	65.23	65.44	70.65	70.21	70.43
	JOINT10	**66.82**	**67.44**†	**67.13**	**71.47**	**72.13***	**71.80**
test	MATE	63.16	62.25	62.70	67.52	66.55	67.03
	TURBO	63.06	62.16	62.61	67.27	66.31	66.79
	PIPELINE	63.63	62.72	63.17	67.62	66.65	67.14
	JOINT10	**63.81**	**63.89**†	**63.85**	**67.79**	**67.88**†	**67.84**

Table 3: Statistically significant differences between the joint system and the pipeline system are marked with † ($p < 0.01$) and $*$ ($p < 0.05$). Significance testing was performed using the Wilcoxon Signed Rank Test (not for F1).

As described in Section 4.4, we cannot evaluate the syntax entirely independently on Hebrew, but we can eliminate the morphological level. Table 4 shows the results of the evaluation when only syntax and surface forms are matched. The overall picture compared to the evaluation shown in Table 3 does not change, however. Also when disregarding the quality of morphology, the JOINT model outperforms the PIPELINE, notably with respect to recall.

6 Related Work

Graph-based Parsing. Our basic architecture resembles the joint constituency parsing and POS tagging model by Rush et al. (2010), but our model

		LABELED			UNLABELED		
data	system	PREC	REC	F1	PREC	REC	F1
dev	MATE	68.05	67.62	67.83	74.70	74.24	74.47
	TURBO	67.97	67.54	67.75	74.58	74.12	74.35
	PIPELINE	68.56	68.14	68.35	74.84	74.37	74.60
	JOINT10	**69.23**	**69.87**†	**69.55**	**74.88**	**75.58**†	**75.23**
test	MATE	66.17	65.22	65.69	71.62	70.60	71.11
	TURBO	66.14	65.19	65.66	71.38	70.35	70.86
	PIPELINE	**66.81**	65.85	66.33	**71.82**	70.79	71.30
	JOINT10	66.63	**66.72**†	**66.68**	71.48	**71.57**†	**71.52**

Table 4: Parsing results for Hebrew, evaluated without morphology. Statistically significant differences between the joint system and the pipeline system are marked with †. Significance testing was performed using the Wilcoxon Signed Rank Test with $p < 0.01$ (not for F1).

needs additional constraints to enforce agreement between the two tasks. Martins et al. (2011a) and Martins et al. (2015) show how such first-order logic constraints can be represented as subproblems in dual decomposition. Similar approaches, where such constraints are used to ensure certain properties in the output structures, have been used e.g. in semantic parsing (Das et al., 2012), compressive summarization (Almeida and Martins, 2013), and joint quotation attribution and coreference resolution (Almeida et al., 2014). Parsers that use dual decomposition are proposed in Koo et al. (2010) and Martins et al. (2010). From Koo et al. (2010), we adopted the idea of using the Chu-Liu-Edmonds algorithm to ensure tree properties in the output as well as second-order parsing with head automata.

Li et al. (2011) extend several higher-order variants of the Eisner decoder (Eisner, 1997) such that POS tags are predicted jointly with syntax. The complexity of their joint models increases by polynomials of the tag set size. Due to the dual decomposition approach, the complexity of our parser stays equal to the complexity of the most complex subproblem, which is the second-order head automata in our case.

Transition-based Parsing. Joint models in transition-based parsing usually introduce a variant of the shift transition that performs the additional task, e.g. it additionally predicts the POS tag and possibly morphological features of a token that is being shifted (Hatori et al., 2011; Bohnet and Nivre, 2012; Bohnet et al., 2013). Optimization over the joint model is achieved by beam search. To also solve the word segmentation task, several models for Chinese were proposed that parse on the level of single characters, forming words from characters with a special append transition (Hatori et al., 2012; Li and Zhou, 2012) or predicting word internal structure along with syntax (Zhang et al., 2014). To use such a transition-based system for the segmentation task in Turkish or Hebrew, the shift transition would have to be changed to do the opposite of the append transition in the Chinese parsers: segment an incoming token into several ones, for example based on the output of a morphological analyzer.

Easy-first Parsing. Ma et al. (2012) introduce a variant of the easy-first parser (Goldberg and Elhadad, 2010a) that uses an additional operation to POS tag input tokens. The operations are ordered such that the parser can only introduce a dependency arc between two tokens that have received a POS tag already. Tratz (2013) presents a similar system for Arabic that defines several more operations to deal with segmentation ambiguity.

Sampling-based Parsing. Zhang et al. (2015) present a joint model that relies on sampling and greedy hill-climbing for decoding, but allows for arbitrarily complex scoring functions thus opening access to global and cross-level features. Such features could be simulated in our model by adding additional factors in the form of soft constraints (constraints with output, see Martins et al. (2015)), but this would introduce a considerable number of additional factors with a notable impact on performance.

Constituency Parsing. Joint models have also been investigated in constituency parsing, notably for Hebrew. Tsarfaty (2006) already discusses full joint models, but the first full parsers were presented in Cohen and Smith (2007), Goldberg and Tsarfaty (2008), and later Goldberg and Elhadad (2013). Green and Manning (2010) present a similar parser for Arabic. Among these, some authors emphasize the importance of including scores from the morphological model into the parsing model, whereas other models do not use them at all. In our parser, the model is trained jointly for both tasks without weighting the two tasks differently.

Parsing Hebrew and Turkish. Joint models for Hebrew parsing were mostly investigated for constituency parsing (see above). There has been some work specifically on Hebrew dependency parsing (Goldberg and Elhadad, 2009; Goldberg and Elhadad, 2010b; Goldberg, 2011), but not in the context of joint models.

Turkish dependency parsing was pioneered in Eryiğit and Oflazer (2006) and Eryiğit et al. (2008). They compare parsing based on inflectional groups to word-based parsing and conclude that the former is more suitable for Turkish. Çetinoğlu and Kuhn (2013) are first to discuss joint models for Turkish and present experiments for joint POS tagging and parsing, but use a pipeline to decide on segmentation and morphological features. To the best of our knowledge, there currently exists no work on full lattice parsing for Turkish.

7 Conclusion

Morphologically rich languages pose many challenges to standard dependency parsing systems, one of them being that the number of tokens in the output is not always known beforehand. Solving this problem in a pipeline setup leads to efficient systems but systematically excludes interaction between the lexical, morphological, and syntactic level of analysis.

In this work, we have presented a graph-based lattice dependency parser that operates on morphological lattices and simultaneously predicts a dependency tree and a path through the lattice. We tested the joint model on the Turkish treebank and the treebank of Modern Hebrew and demonstrated that the joint model outperforms three state-of-the-art pipeline models. We presented the first results for full lattice parsing on the Turkish treebank. The results on the Hebrew treebank corroborate findings in constituency parsing (Cohen and Smith, 2007; Goldberg and Tsarfaty, 2008).

Acknowledgments

We thank our anonymous reviewers for their helpful comments. We also thank Anders Björkelund for many useful discussions. This work was funded by the Deutsche Forschungsgemeinschaft (DFG) via SFB 732, projects D2 and D8.

References

Miguel Almeida and Andre Martins. 2013. Fast and Robust Compressive Summarization with Dual Decomposition and Multi-Task Learning. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 196–206, Sofia, Bulgaria, August. Association for Computational Linguistics.

Mariana S. C. Almeida, Miguel B. Almeida, and André F. T. Martins. 2014. A Joint Model for Quotation Attribution and Coreference Resolution. In *Proceedings of the 14th Conference of the European Chapter of the Association for Computational Linguistics*, pages 39–48, Gothenburg, Sweden, April. Association for Computational Linguistics.

Bernd Bohnet and Joakim Nivre. 2012. A Transition-Based System for Joint Part-of-Speech Tagging and Labeled Non-Projective Dependency Parsing. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 1455–1465, Jeju, South Korea. Association for Computational Linguistics.

Bernd Bohnet, Joakim Nivre, Igor Boguslavsky, Richrd Farkas, Filip Ginter, and Jan Haji. 2013. Joint Morphological and Syntactic Analysis for Richly Inflected Languages. *Transactions of the Association for Computational Linguistics*, 1:415–428.

Bernd Bohnet. 2009. Efficient Parsing of Syntactic and Semantic Dependency Structures. In *Proceedings of the Thirteenth Conference on Computational Natural Language Learning (CoNLL 2009): Shared Task*, pages 67–72, Boulder, Colorado, June. Association for Computational Linguistics.

Bernd Bohnet. 2010. Very high accuracy and fast dependency parsing is not a contradiction. In *Proceedings of the 23rd International Conference on Computational Linguistics*, pages 89–97, Beijing, China. International Committee on Computational Linguistics.

Xavier Carreras. 2007. Experiments with a Higher-Order Projective Dependency Parser. In *Proceedings of the CoNLL Shared Task Session of EMNLP-CoNLL 2007*, pages 957–961, Prague, Czech Republic, June. Association for Computational Linguistics.

Özlem Çetinoğlu and Jonas Kuhn. 2013. Towards Joint Morphological Analysis and Dependency Parsing of Turkish. In *Proceedings of the Second International Conference on Dependency Linguistics (DepLing 2013)*, pages 23–32, Prague, Czech Republic, August. Charles University in Prague, Matfyzpress, Prague, Czech Republic.

Özlem Çetinoğlu. 2014. Turkish Treebank as a Gold Standard for Morphological Disambiguation and Its Influence on Parsing. In Nicoletta Calzolari (Conference Chair), Khalid Choukri, Thierry Declerck, Hrafn Loftsson, Bente Maegaard, Joseph Mariani, Asuncion Moreno, Jan Odijk, and Stelios Piperidis, editors, *Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC'14)*, Reykjavik, Iceland, may. European Language Resources Association (ELRA).

Yoeng-Jin Chu and Tseng-Hong Liu. 1965. On the Shortest Arborescence of a Directed Graph. *Scientia Sinica*, 14(10):1396–1400.

Shay B. Cohen and Noah A. Smith. 2007. Joint morphological and syntactic disambiguation. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 208–217, Prague, Czech Republic. Association for Computational Linguistics.

Koby Crammer, Ofer Dekel, Shai Shalev-Shwartz, and Yoram Singer. 2003. Online passive-aggressive algorithms. In *Proceedings of the 16th Annual Conference on Neural Information Processing Systems*, volume 7, pages 1217–1224, Cambridge, Massachusetts, USA. MIT Press.

Dipanjan Das, André F. T. Martins, and Noah A. Smith. 2012. An Exact Dual Decomposition Algorithm for Shallow Semantic Parsing with Constraints. In **SEM 2012: The First Joint Conference on Lexical and Computational Semantics – Volume 1: Proceedings of the main conference and the shared task, and Volume 2: Proceedings of the Sixth International Workshop on Semantic Evaluation (SemEval 2012)*, pages 209–217, Montréal, Canada, 7-8 June. Association for Computational Linguistics.

Jack Edmonds. 1967. Optimum Branchings. *Journal of Research of the National Bureau of Standards*, 71B(4):233–240.

Jason Eisner. 1997. Bilexical Grammars and a Cubic-Time Probabilistic Parser. In *Proceedings of the 5th International Workshop on Parsing Technologies (IWPT)*, pages 54–65, MIT, Cambridge, MA, sep.

Gülşen Eryiğit and Kemal Oflazer. 2006. Statistical dependency parsing of Turkish. In *Proceedings of the 11th Conference of the European Chapter of the Association for Computational Linguistics*, pages 89–96, Trento, Italy. Association for Computational Linguistics.

Gülşen Eryiğit, Joakim Nivre, and Kemal Oflazer. 2008. Dependency Parsing of Turkish. *Computational Linguistics*, 34(3):357–389.

Gülşen Eryiğit, Tugay Ilbay, and Ozan Arkan Can. 2011. Multiword Expressions in Statistical Dependency Parsing. In *Proc. of the SPMRL Workshop of IWPT*, pages 45–55, Dublin, Ireland.

Gülşen Eryiğit. 2012. The Impact of Automatic Morphological Analysis & Disambiguation on Dependency Parsing of Turkish. In Nicoletta Calzolari, Khalid Choukri, Thierry Declerck, Mehmet Uğur Doğan, Bente Maegaard, Joseph Mariani, Jan Odijk, and Stelios Piperidis, editors, *Proceedings of the Eighth International Conference on Language Resources and Evaluation (LREC-2012)*, pages 1960–1965, Istanbul, Turkey, May. European Language Resources Association (ELRA). ACL Anthology Identifier: L12-1056.

Yoav Freund and Robert E. Schapire. 1999. Large margin classification using the perceptron algorithm. *Machine Learning*, 37(3):277–296.

Yoav Goldberg and Michael Elhadad. 2009. Hebrew Dependency Parsing: Initial Results. In *Proceedings of the 11th International Conference on Parsing Technologies (IWPT'09)*, pages 129–133, Paris, France, October. Association for Computational Linguistics.

Yoav Goldberg and Michael Elhadad. 2010a. 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*, pages 742–750, Los Angeles, California, June. Association for Computational Linguistics.

Yoav Goldberg and Michael Elhadad. 2010b. Easy-First Dependency Parsing of Modern Hebrew. In *Proceedings of the NAACL HLT 2010 First Workshop on Statistical Parsing of Morphologically-Rich Languages*, pages 103–107, Los Angeles, CA, USA, June. Association for Computational Linguistics.

Yoav Goldberg and Michael Elhadad. 2013. Word segmentation, unknown-word resolution, and morphological agreement in a hebrew parsing system. *Computational Linguistics*, 39(1):121–160.

Yoav Goldberg and Reut Tsarfaty. 2008. A single generative model for joint morphological segmentation and syntactic parsing. In *Proceedings of the 46th Annual Meeting of the Association for Computational Linguistics*, pages 371–379, Columbus, Ohio. Association for Computational Linguistics.

Yoav Goldberg. 2011. *Automatic Syntactic Processing of Modern Hebrew*. Ph.D. thesis, Ben Gurion University, Beer Sheva, Israel.

Spence Green and Christopher D. Manning. 2010. Better Arabic Parsing: Baselines, Evaluations, and Analysis. In *Proceedings of the 23rd International Conference on Computational Linguistics (Coling 2010)*, pages 394–402, Beijing, China, August. Coling 2010 Organizing Committee.

Jun Hatori, Takuya Matsuzaki, Yusuke Miyao, and Jun'ichi Tsujii. 2011. Incremental Joint POS Tagging and Dependency Parsing in Chinese. In *Proceedings of 5th International Joint Conference on Natural Language Processing*, pages 1216–1224, Chiang Mai, Thailand, November. Asian Federation of Natural Language Processing.

Jun Hatori, Takuya Matsuzaki, Yusuke Miyao, and Jun'ichi Tsujii. 2012. Incremental Joint Approach to Word Segmentation, POS Tagging, and Dependency Parsing in Chinese. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1045–1053, Jeju Island, Korea, July. Association for Computational Linguistics.

Terry Koo, Alexander M. Rush, Michael Collins, Tommi Jaakkola, and David Sontag. 2010. Dual Decomposition for Parsing with Non-Projective Head Automata. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, pages 1288–1298, Cambridge, MA, October. Association for Computational Linguistics.

Zhongguo Li and Guodong Zhou. 2012. Unified Dependency Parsing of Chinese Morphological and Syntactic Structures. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 1445–1454, Jeju Island, Korea, July. Association for Computational Linguistics.

Zhenghua Li, Min Zhang, Wanxiang Che, Ting Liu, Wenliang Chen, and Haizhou Li. 2011. Joint Models for Chinese POS Tagging and Dependency Parsing. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing*, pages 1180–1191, Edinburgh, Scotland, UK, July. Association for Computational Linguistics.

Ji Ma, Tong Xiao, Jingbo Zhu, and Feiliang Ren. 2012. Easy-First Chinese POS Tagging and Dependency Parsing. In *Proceedings of COLING 2012*, pages 1731–1746, Mumbai, India, December. The COLING 2012 Organizing Committee.

Andre Martins, Noah Smith, and Eric Xing. 2009. Concise Integer Linear Programming Formulations for Dependency Parsing. 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 342–350, Suntec, Singapore, August. Association for Computational Linguistics.

Andre Martins, Noah Smith, Eric Xing, Pedro Aguiar, and Mario Figueiredo. 2010. Turbo Parsers: Dependency Parsing by Approximate Variational Inference. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, pages 34–44, Cambridge, MA, October. Association for Computational Linguistics.

Andre Martins, Mario Figueiredo, Pedro Aguiar, Noah Smith, and Eric Xing. 2011a. An Augmented Lagrangian Approach to Constrained MAP Inference. In Lise Getoor and Tobias Scheffer, editors, *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, ICML '11, pages 169–176, New York, NY, USA, June. ACM.

Andre Martins, Noah Smith, Mario Figueiredo, and Pedro Aguiar. 2011b. Dual Decomposition with Many Overlapping Components. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing*, pages 238–249, Edinburgh, Scotland, UK., July. Association for Computational Linguistics.

Andre Martins, Miguel Almeida, and Noah A. Smith. 2013. Turning on the Turbo: Fast Third-Order Non-Projective Turbo Parsers. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 617–622, Sofia, Bulgaria, August. Association for Computational Linguistics.

André F. T. Martins, Mário A. T. Figueiredo, Pedro M. Q. Aguiar, Noah A. Smith, and Eric P. Xing. 2015. AD3: Alternating Directions Dual Decomposition for MAP Inference in Graphical Models. *Journal of Machine Learning Research*, 16:495–545.

Ryan McDonald and Fernando Pereira. 2006. Online learning of approximate dependency parsing algorithms. In *Proceedings of the 11th Conference of the European Chapter of the Association for Computational Linguistics*, pages 81–88, Trento, Italy. Association for Computational Linguistics.

Ryan McDonald, Fernando Pereira, Kiril Ribarov, and Jan Hajic. 2005. Non-Projective Dependency Parsing using Spanning Tree Algorithms. In *Proceedings of Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing*, pages 523–530, Vancouver, British Columbia, Canada, October. Association for Computational Linguistics.

Saul B. Needleman and Christian D. Wunsch. 1970. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of molecular biology*, 48(3):443–453.

Kemal Oflazer, Bilge Say, Dilek Zeynep Hakkani-Tür, and Gökhan Tür. 2003. Building a Turkish Treebank. In Anne Abeille, editor, *Building and Exploiting Syntactically-annotated Corpora*. Kluwer Academic Publishers, Dordrecht.

Kemal Oflazer. 1994. Two-level Description of Turkish Morphology. *Literary and Linguistic Computing*, 9(2):137–148.

Alexander M. Rush, David Sontag, Michael Collins, and Tommi Jaakkola. 2010. On Dual Decomposition and Linear Programming Relaxations for Natural Language Processing. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, pages 1–11, Cambridge, MA, October. Association for Computational Linguistics.

Haşim Sak, Tunga Güngör, and Murat Saraçlar. 2008. Turkish Language Resources: Morphological Parser, Morphological Disambiguator and Web Corpus. In *Proc. of GoTAL 2008*, pages 417–427.

Djamé Seddah, Reut Tsarfaty, Sandra Kübler, Marie Candito, Jinho D. Choi, Richárd Farkas, Jennifer Foster, Iakes Goenaga, Koldo Gojenola Galletebeitia, Yoav Goldberg, Spence Green, Nizar Habash, Marco Kuhlmann, Wolfgang Maier, Joakim Nivre, Adam Przepiórkowski, Ryan Roth, Wolfgang Seeker, Yannick Versley, Veronika Vincze, Marcin Woliński, Alina Wróblewska, and Eric Villemonte de la Clergerie. 2013. Overview of the SPMRL 2013 Shared Task: A Cross-Framework Evaluation of Parsing Morphologically Rich Languages. In *Proceedings of the Fourth Workshop on Statistical Parsing of Morphologically-Rich Languages*, pages 146–182, Seattle, Washington, USA, October. Association for Computational Linguistics.

Djamé Seddah, Sandra Kübler, and Reut Tsarfaty. 2014. Introducing the SPMRL 2014 Shared Task on Parsing Morphologically-rich Languages. In *Proceedings of the First Joint Workshop on Statistical Parsing of Morphologically Rich Languages and Syntactic Analysis of Non-Canonical Languages*, pages 103–109, Dublin, Ireland, August. Dublin City University.

Khalil Sima'an, Alon Itai, Yoad Winter, Alon Altman, and Noa Nativ. 2001. Building a tree-bank of modern Hebrew text. *Traitement Automatique des Langues*, 42(2):247–380.

Ben Taskar, Vassil Chatalbashev, Daphne Koller, and Carlos Guestrin. 2005. Learning Structured Prediction Models: A Large Margin Approach. In *Proceedings of the 22th Annual International Conference on Machine Learning*, pages 896–903, Bonn, Germany. ACM.

Stephen Tratz. 2013. A Cross-Task Flexible Transition Model for Arabic Tokenization, Affix Detection, Affix Labeling, POS Tagging, and Dependency Parsing. In *Proceedings of the Fourth Workshop on Statistical Parsing of Morphologically-Rich Languages*, pages 34–45, Seattle, Washington, USA, October. Association for Computational Linguistics.

Reut Tsarfaty, Djamé Seddah, Yoav Goldberg, Sandra Kuebler, Yannick Versley, Marie Candito, Jennifer Foster, Ines Rehbein, and Lamia Tounsi. 2010. Statistical Parsing of Morphologically Rich Languages (SPMRL) What, How and Whither. In *Proceedings of the NAACL HLT 2010 First Workshop on Statistical Parsing of Morphologically-Rich Languages*, pages 1–

12, Los Angeles, CA, USA, June. Association for Computational Linguistics.

Reut Tsarfaty, Joakim Nivre, and Evelina Andersson. 2012. Joint Evaluation of Morphological Segmentation and Syntactic Parsing. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 6–10, Jeju Island, Korea, July. Association for Computational Linguistics.

Reut Tsarfaty. 2006. Integrated Morphological and Syntactic Disambiguation for Modern Hebrew. In *Proceedings of the COLING/ACL 2006 Student Research Workshop*, pages 49–54, Sydney, Australia, July. Association for Computational Linguistics.

Meishan Zhang, Yue Zhang, Wanxiang Che, and Ting Liu. 2014. Character-Level Chinese Dependency Parsing. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1326–1336, Baltimore, Maryland, June. Association for Computational Linguistics.

Yuan Zhang, Chengtao Li, Regina Barzilay, and Kareem Darwish. 2015. Randomized Greedy Inference for Joint Segmentation, POS Tagging and Dependency Parsing. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 42–52, Denver, Colorado, May–June. Association for Computational Linguistics.