Turkish Language Resources: Morphological Parser, Morphological Disambiguator and Web Corpus

Haşim Sak[1], Tunga Güngör[1], and Murat Saraçlar[2]

[1] Boğaziçi University, Computer Engineering Department, Bebek, 34342 İstanbul, Turkey
hasim.sak@boun.edu.tr, gungort@boun.edu.tr
[2] Boğaziçi University, Electrical and Electronic Engineering Department, Bebek, 34342 İstanbul, Turkey
murat.saraclar@boun.edu.tr

Abstract. In this paper, we propose a set of language resources for building Turkish language processing applications. Specifically, we present a finite-state implementation of a morphological parser, an averaged perceptron-based morphological disambiguator, and compilation of a web corpus. Turkish is an agglutinative language with a highly productive inflectional and derivational morphology. We present an implementation of a morphological parser based on two-level morphology. This parser is one of the most complete parsers for Turkish and it runs independent of any other external system such as PC-KIMMO in contrast to existing parsers. Due to complex phonology and morphology of Turkish, parsing introduces some ambiguous parses. We developed a morphological disambiguator with accuracy of about 98% using averaged perceptron algorithm. We also present our efforts to build a Turkish web corpus of about 423 million words.

Keywords: Morphological parsing, Morphological disambiguation, Turkish, Web corpus.

1 Finite-State Morphological Parser

Morphological parsing is the problem of breaking a word such as *çocuklar* (children) into the constituent morphemes, *çocuk* (child) and *-lar* (plural suffix). To build a morphological parser, we need three components: a lexicon listing the stem words annotated with some information such as part-of-speech of the words to determine which morphological rules apply to them, a morphotactics component (morphosyntax) that describes the word formation by specifying the ordering of morphemes, and a morphophonemics component that describes the phonological alternations occurring in the morphemes during word formation. In finite-state morphology, all these components can be implemented using finite-state transducers.

To implement phonological rules, we used the two-level morphology formalism of Koskenniemi [5]. Two-level morphology is a formalism for describing morphological alternations. In this formalism, the phonological rules denote regular relations that can

A. Ranta (Ed.): GoTAL 2008, LNAI 5221, pp. 417–427, 2008.
© Springer-Verlag Berlin Heidelberg 2008

be represented by finite-state transducers. Two-level rules are applied in parallel or when implemented as finite-state transducers they can be intersected to a single morphophonemics transducer.

To show how two-level phonology is used to model phonological phenomena, we give an example for vowel harmony in Turkish [6]. In Turkish, the /a/ vowel in suffixes is realized as /a/ or /e/ in surface form depending on the word they are attached to. According to vowel harmony, the /a/ vowel changes its form to agree in backness with the preceding stem vowel. A two-level rule that describes this phenomena in the case of front vowels is given below.

A:e ⇒ @:FV [@:CONS | @:ε]* _

In this rule, "A" symbol is used for lexical representation of /a/ vowel in suffixes. "FV" symbol represents the front vowels /e/, /i/, /ö/, and /ü/. "CONS" symbol represents the set of consonants. "@" symbol means any symbol in the alphabet. This rule states that /a/ vowel (/A/ in lexical form) may be converted to /e/ vowel only if it is preceded with a surface front vowel followed possibly by a number of symbols having consonants and epsilon realizations in the surface form. The finite-state transducer realization for this rule is shown in Figure 1.

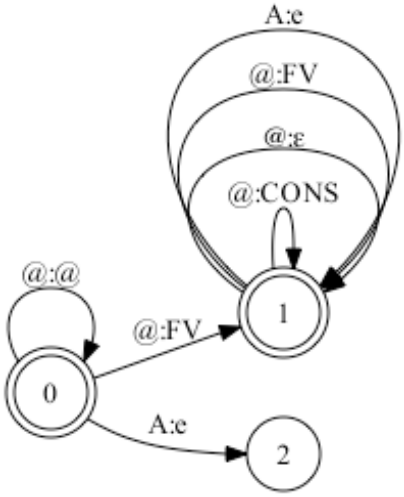

Fig. 1. Transducer for Turkish vowel harmony: "@:@" symbol represents any feasible lexical/surface pair absent in the transducer. "@" symbol represents any other symbol that is not used on any arc.

The lexicon and morphotactics can also be encoded into a single finite-state transducer as shown in Figure 2. This FST implements a simple nominal inflection for Turkish. The input side of this transducer encodes the morphological features to be returned as the morphological parse of the words. The output side is meant to be input to the phonological rules transducer, therefore it needs to be expanded to letter sequences. As you can see the output morphemes are marked with special symbols to encode phonological alternations in the rules transducer.

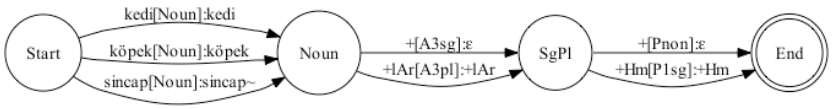

Fig. 2. A transducer for a simple Turkish nominal inflection

Given the morphophonemics and lexicon/morphotactics transducers, it is quite easy to build a transducer that implements a morphological parser. Simply, we compose the lexicon/morphotactics transducer with the morphophonemics transducer, then invert the resulting transducer to do morphological analysis rather than generation.

In this implementation, we aimed to build a morphological parser that is not dependent on any external system for running. We wanted to construct a finite-state transducer that implements a Turkish morphological parser and that can be embedded in other NLP applications. Therefore in this implementation, we did not use external systems such as PC-KIMMO and Xerox finite-state tools. For finite-state operations we used AT&T FSM tools [8], but these tools are not required for the parser to run.

We compiled a new lexicon of 54,267 root words. To compile this lexicon and to ensure the correct spelling of the words we used the Turkish Language Institution (TDK) dictionary.

An example output from the morphological parser for the word *alın* is given below:

alın[Noun]+[A3sg]+[Pnon]+[Nom]
al[Noun]+[A3sg]+Hn[P2sg]+[Nom]
al[Adj]-[Noun]+[A3sg]+Hn[P2sg]+[Nom]
al[Noun]+[A3sg]+[Pnon]+NHn[Gen]
al[Adj]-[Noun]+[A3sg]+[Pnon]+NHn[Gen]
alın[Verb]+[Pos]+[Imp]+[A2sg]
al[Verb]+[Pos]+[Imp]+YHn[A2pl]
al[Verb]-Hn[Verb+Pass]+[Pos]+[Imp]+[A2sg]

In the morphological parse output the first part is always the root word. Then the part-of-speech tag for the stem is given in brackets. These are followed by a set of lexical morphemes with the associated morphological features. The inflectional morphemes start with a + sign, and the derivational morphemes start with a - sign. The morphological features are given in brackets. If the morpheme is a derivational one, then the morphological features for that morpheme start with the part-of-speech of the derived word form. It is also possible that morphological features can be assigned in the absence of morphemes.

2 Morphological Disambiguation

A morphological parser for a language with complex morphology may return more than one possible analysis of a word. The ambiguous parses of an example word *alın* were shown in the previous section. As can be seen in that example, some of the parses have different root words and have unrelated morphological features due to the productive

morphology of Turkish. This morphological ambiguity needs to be resolved for further language processing. Several approaches have been proposed for morphosyntactic tagging in inflective and agglutinative languages, e.g. [2,3,4,7,9,10,13].

An application of the averaged perceptron algorithm to the morphological disambiguation of Turkish text is described in [10]. In that study, a baseline trigram-based model of [2] is used to enumerate n-best candidates of alternative morphological parses of a sentence. Then the averaged perceptron algorithm is applied to re-rank the n-best candidate list using a set of features. In this study, we do not use a baseline model to generate n-best candidates. Instead, we do a Viterbi decoding of the best path in the network of ambiguous morphological parses of the words in a sentence using the averaged perceptron algorithm to train model parameters as explained in the next section.

The set of features that we included in the model are the same as in [10]. This feature set takes into account the current morphosyntactic tag (parse) and the history of the previous two tags. Therefore, we can do a left to right Viterbi decoding for the best morphological parse sequence for a sentence.

2.1 Perceptron Algorithm

A variant of the perceptron algorithm that can be applied to problems such as tagging and parsing is given in Figure 3. The algorithm estimates a parameter vector $\overline{\alpha}$ that can be used for mapping from inputs $x \in X$ to outputs $y \in Y$ using a set of training examples (x_i, y_i). In our setting, X is a set of sentences and Y is a set of possible morphological parse sequences. The algorithm makes multiple passes (denoted by T) over the training examples. For each example, it finds the highest scoring candidate among all candidates using the current parameter values. If the highest scoring candidate is not the correct one, it updates the parameter vector $\overline{\alpha}$ by the difference of the feature vector representation of the correct candidate and the highest scoring candidate. This way of parameter update increases the parameter values for features in the correct candidate and decreases parameter values for features in the competitor.

This algorithm can be set up for the morphological disambiguation problem as follows:

- The training examples are the sentence $x_i = w^i_{[1:n_i]}$ and the morphological parse sequence $y_i = t^i_{[1:n_i]}$ pairs for $i = 1,..,n$, where n is the number of training sentences and n_i is the length of the i'th sentence.

Inputs: Training examples (x_i, y_i)
Initialization: Set $\overline{\alpha} = 0$
Algorithm:
For $t = 1,..,T$, $i = 1,..,n$
Calculate $z_i = \arg\max_{z \in GEN(x_i)} \Phi(x_i, z) \cdot \overline{\alpha}$
If $(z_i \neq y_i)$ **then** $\overline{\alpha} = \overline{\alpha} + \Phi(x_i, y_i) - \Phi(x_i, z_i)$
Output: Parameters $\overline{\alpha}$

Fig. 3. A variant of the perceptron algorithm

- The function $GEN(x_i)$ maps the input sentence to the candidate parse sequences.
- The representation $\Phi(x,y) \in \Re^d$ is a feature vector, the components of which are defined as $\Phi_s\left(w_{[1:n]}, t_{[1:n]}\right) = \sum_{i=1}^{n} \phi_s(t_{i-2}, t_{i-1}, t_i)$, where $\phi_s(t_{i-2}, t_{i-1}, t_i)$ is an indicator function for a feature that depends on the current morphosyntactic tag (morphological parse) and the history of the previous two tags. Then the feature vector components $\Phi_s\left(w_{[1:n_i]}, t_{[1:n_i]}\right)$ are just the counts of the local features $\phi_s(t_{i-2}, t_{i-1}, t_i)$. For example one feature might be:

$$\phi_{100}(t_{i-2}, t_{i-1}, t_i) = \begin{cases} 1 \text{ if current parse } t_i \text{ is al} + \text{Verb} + \text{Pos} + \text{Imp} + \text{A2pl} \\ \quad \text{and previous parse } t_{i-1} \text{ is a pronoun} \\ 0 \text{ otherwise} \end{cases}$$

- The expression $\Phi(x,y) \cdot \overline{\alpha}$ is the inner product $\sum_s \alpha_s \Phi_s(x,y)$.
- The function $\arg\max_{z \in GEN(x_i)} \Phi(x_i, z) \cdot \overline{\alpha}$ can be efficiently calculated using dynamic programming since the features that we use depend on only the current tag and the previous two tags.

For the application of the model to the test examples, we use the "averaged parameters" since they are more robust to noisy or unseparable data [1]. The averaged parameters γ are calculated by summing the parameter values for each feature after each training example and dividing this sum by the total number of examples used to update the parameters. With this setting, the perceptron algorithm learns an averaged parameter vector γ that can be used to choose the most likely morphological parse sequence of a test sentence x using the following function:

$$\begin{aligned} F(x) &= \arg\max_{y \in GEN(x)} \Phi(x,y) \cdot \gamma \\ &= \arg\max_{y \in GEN(x)} \sum_{s=1}^{d} \Phi_s(x,y) \cdot \gamma_s \end{aligned}$$

2.2 Experiments

We used a morphologically disambiguated Turkish corpus of about 950,000 tokens (including markers such as begin and end of sentence markers). Alternative ambiguous parses of the words are also available in the corpus as output from a morphological analyzer. This data set was divided into a training, development, and test set. The training set size is about 750,000 tokens or 45,000 sentences. The development set size is about 40,000 tokens or 2,500 sentences. The test set size is also about 40,000 tokens or 2,500 sentences. The training set is used for parameter estimation and the development set is used to tune some of the parameters in the perceptron algorithm. The final tests were done on the test set.

The accuracy of the perceptron algorithm on the test set is 97.81%. For a comparison of accuracy of the Viterbi decoding with averaged perceptron with the trigram-based model of [2] and trigram-based model plus perceptron re-ranking as described in [10], see Table 1.

Table 1. Comparative Results on Test Set (40K tokens)

Method	Accuracy (%)
Trigram-based model [2]	93.61
Trigram-based + Perceptron [10]	96.76
Perceptron (this study)	97.81

3 Web Corpus

In the domain of language processing, we need large corpora for the application and evaluation of statistical methods. Such corpora are also important for empirical methods that the linguists and lexicographers use to infer information about language. There have been very few efforts to build a Turkish text corpus [11,12] and they were quite limited in terms of size and coverage to be successfully used in statistical natural language applications.

In this research, a large corpus for Turkish was built and cleaned using some heuristics and the morphological parser. The corpus is composed of four sub corpora. Three of these corpora (referred as *NewsCor*) are from three major newspapers in Turkish. The other corpus (referred as *GenCor*) is a general sampling of Turkish web pages. The combined corpus of these two corpora will be referred as *BOUN Corpus*.

For data collection from the web, we implemented a web crawler - an automated script to browse the web as used by the search engines. Since the collected data from the web is very noisy, we employed some automatic normalization and filtering methods to clean the corpus. We followed a multi step process to clean the corpus as described below:

1. Decode HTML entities
2. Trim white spaces at the start and end of the lines
3. Estimate letter sequence counts from a Turkish text and use these counts to filter documents
4. Remove duplicate lines to get rid of repetitions in web pages, such as text in navigation menus
5. Remove documents with less than 1,000 characters
6. Parse the documents using the morphological parser and remove those for which more than 25% of the words cannot be parsed

The normalization and filtering step removes about 60% of the text collected for *NewsCor* and 90% of the text collected for *GenCor*. This difference is expected since the web corpus data is very noisy when compared to the newspaper data.

The tokenization and segmentation of the corpus is often needed in language applications. Since the corpus is very large for manual operation, we employed automatic methods to tokenize and segment the corpus to sentences. The morphological parser that we have developed was very useful in this process. We used the parser as a computational lexicon to look for the words in the corpus.

For the encoding of the web corpus, we used the XML Corpus Encoding Standard, XCES (see http://xces.org) as used in [12]. We encode the corpus in paragraph and sentence level. We also plan to annotate the corpus linguistically in morphosyntactic level.

3.1 Contents of the Corpus

As stated before, Turkish web corpus is formed of four sub corpora. Three of these are from three major newspapers in Turkish and the other one is a general sampling of Turkish web pages. The statistics about the number of words (all words in the corpus), number of tokens (words and lexical units such as punctuation marks), and types (distinct tokens) are shown in Table 2. The percentages of tokens and types that can be successfully parsed by the morphological parser are also indicated. We can interpret the figures on the table from different points of view.

First, we observe that, due to the agglutinative nature of the language, the number of types is quite large. Turkish dictionaries on general domain have a typical size of 50,000-100,000 words. The number of types in the corpus being about 50-60 times larger than the typical number of (mostly) stems indicates that derived words are used commonly in written language.

Table 2. Web Corpus Size

Corpus	Words	Tokens	Types	Tokens parsed (%)	Types parsed (%)
Milliyet	59M	68M	1.1M	96.7	63.5
Ntvmsbnc	75M	86M	1.2M	96.4	55.8
Radikal	50M	58M	1.0M	97.0	65.7
NewsCor	184M	212M	2.2M	96.7	52.2
GenCor	239M	279M	3.0M	94.6	39.5
BOUN Corpus	423M	491M	4.1M	95.5	38.4

Second, a significant difference exists between the percentages of tokens and types successfully parsed. This is an expected result, since most of the tokens in the corpus are grammatical words and there is a relatively small amount of other kinds of tokens (punctuation symbols, proper nouns, etc.) that cannot be parsed. On the other hand, each distinct token is treated equally in the last column of the table, without taking frequencies into consideration. We see that the parser can return an analysis only for 38.4% of the types; the rest cannot be parsed. However, this percentage of types in fact constitutes 95.5% of the corpus. The main reasons for the unparsed types are the proper nouns that do not exist in the lexicon and the spelling errors in the corpus.

Another observation is about the cleanness of the corpus. When we compare *GenCor* with *NewsCor*, we notice a decrease in the number of words that can be parsed. The difference is about 2% in the case of tokens while it is much higher (12.7%) in the case of types. These figures indicate that *NewsCor* is much cleaner than *GenCor*, as might be expected. Also the analysis of the number of words, tokens, and types in the two subcorpora shows that *GenCor* includes more types that are not actually words and there are also some unparsed tokens with high frequencies on this subcorpus. These observations signal that the words used by general web users are more diverse than those used in news portals and some of these words seem to be accepted (due to their high frequencies) by the web community.

Finally, the performance ratios for the morphological parser are quite satisfactory. The success is 96.7% on *NewsCor* and it is slightly lower for *GenCor* due to special characteristics of the written text on the web.

3.2 Corpus Statistics

In this section, we will present statistical results about the corpus in order to get an idea about the coverage of a corpus of this size for an agglutinative language and also to observe the morphological characteristics of Turkish language. Figure 4 shows statistics about the types relative to the corpus size (number of tokens). As can be seen, the number of types is increasing continuously for both corpora and for the combined corpus. It seems that if corpus size is increased beyond the current size of 491M tokens, new types will still continue to emerge. This is supported by the evidence that when the corpus size was increased from 490M to 491M, 5,539 new types (of which 1,009 can be parsed successfully) have been added to the corpus. This is partly due to the productive morphological structure of Turkish and partly to the rich web environment. These facts indicate that the size of the current corpus does not cover all language usage. It should be extended until at least the number of types that can be parsed becomes stable, corresponding to the situation that nearly all possible derived forms are represented in the corpus. Adding more data beyond this limit will just cause an increase in the number of special tokens (e.g. proper nouns) and misspelled words.

Figure 5 shows coverage statistics with respect to the vocabulary size (number of types). The figure was obtained by first sorting the types in decreasing order of frequencies and then summing up the frequencies beginning from the topmost entry for the indicated vocabulary sizes. 50% of the corpus is formed of only about 1,000 distinct words. We observe that about 300K types are necessary in order to attain an acceptable coverage ratio (97-98%). The agglutinative nature of the language and the diversity of the web contents are the basic reasons of this result. The analysis of a similar statistic for the percentages of infrequent types shows that almost half of the types (about 2.0M) occur only once in the corpus. The number of types occurring less than 10 times is 3.4M and they represent 7.5M tokens in the corpus. Thus, the majority of types in the corpus are very infrequent and 98.4% of the corpus is formed of only 15.9% of the types.

To understand the source of the large number of types in the corpus, we give statistics for the stems and lexical endings (tokens stripped of their stems in lexical form such as +lAr+Hn) of the tokens that can be parsed in Figure 6. As the number of tokens considered reaches to the size of the corpus, the number of unique stems approaches to the size of our lexicon (54,267 root words). However, as can be expected, all the words in the lexicon do not appear in the corpus and even a corpus of this size does not contain any occurrence of 5,630 words. On the other hand, the number of unique endings increases steadily as new data are added. Note that the figure considers only the tokens that can be successfully parsed. Hence, this increase means that people freely derive new word forms by making use of suffix combinations not used before. This is an interesting result. Although we know that theoretically there is no limit on the number of derivations in Turkish, we might expect that in practice a (large) subset of all possible derived forms will cover the daily use of the language.

However, this expectation does not hold even for a set of nearly 500M tokens and about 9,000 new words, 40 stems, and 60 lexical endings emerge per 10M tokens at this size. When the whole corpus is considered, about 30 different words can be obtained from a single root form, which is an indication of the productive morphological structure of the language.

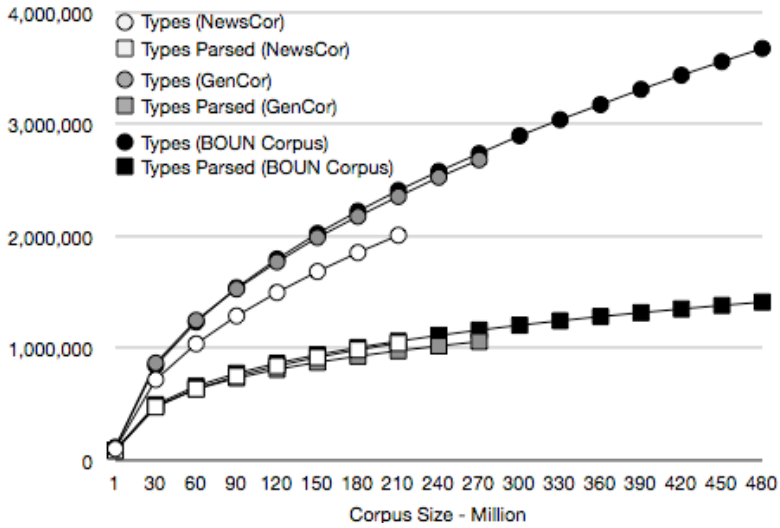

Fig. 4. Type statistics for subcorpora and combined corpus

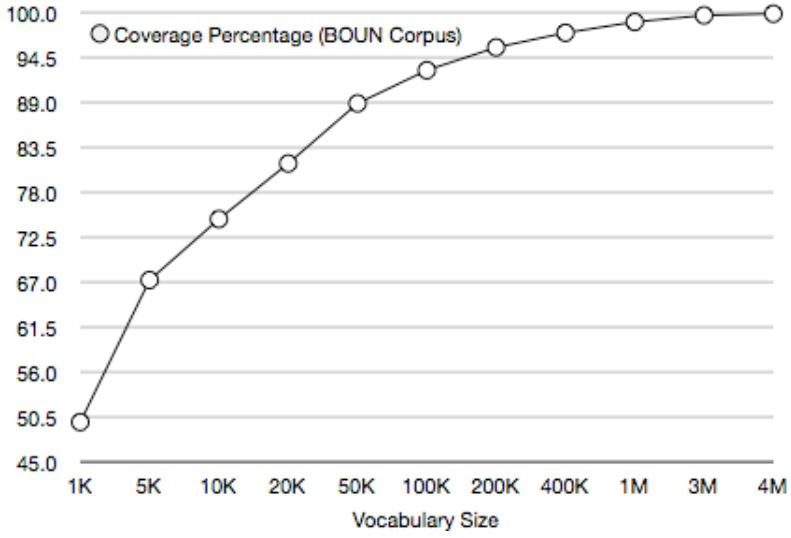

Fig. 5. Coverage statistics for subcorpora and combined corpus

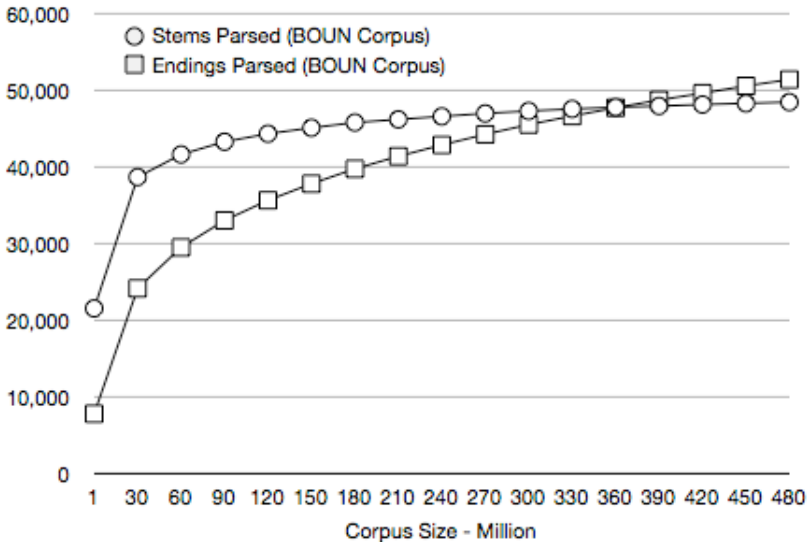

Fig. 6. Stem and ending statistics for subcorpora and combined corpus

4 Conclusions

In this paper, we presented some language resources and tools for Turkish that can be used to build Turkish NLP applications. We already used the morphological parser as a computational lexicon to implement a spell checker for Mac OS X. Our primary motivation in compiling these resources is to develop a large vocabulary speech recognition system for Turkish.

The language resources obtained as output of this research are: (i) A highly efficient finite-state morphological parser that does not depend on any other environment to run. It is one of the most complete parsers in terms of lexicon coverage, morphotactics, and morphophonemics; (ii) An efficient averaged perceptron-based morphological disambiguator that uses Viterbi decoding. The disambiguation accuracy of 97.81% is the highest accuracy reported so far for Turkish; (iii) A web corpus containing about 500 million tokens. The corpus has been cleaned using some heuristics and the morphological parser developed in this work and then converted to XCES XML format.

We believe that the methodologies described here for Turkish can be applied to other languages with complex morphology to build high-quality language resources. The resources obtained have the potential of being used as building blocks in large-scale language applications. As a future work, we plan to use the morphological parser and the disambiguator for linguistic annotation of the web corpus.

Acknowledgements

This work was supported by Boğaziçi University Research Fund under the grant numbers 06A102 and 08M103, and by TÜBİTAK under the grant number 107E261.

References

1. Collins, M.: Discriminative Training Methods for Hidden Markov Models: Theory and Experiments with Perceptron Algorithms. In: EMNLP (2002)
2. Dilek, Z.H.T., Oflazer, K., Tür, G.: Statistical Morphological Disambiguation for Agglutinative Languages. Computers and the Humanities 36(4) (2002)
3. Ezeiza, N., Alegria, I., Arriola, J.M., Urizar, R., Aduriz, I.: Combining Stochastic and Rule-based Methods for Disambiguation in Agglutinative Languages. In: COLING-ACL (1998)
4. Hajic, J., Hladka, B.: Tagging Inflective Languages: Prediction of Morphological Categories for a Rich, Structured Tagset. In: COLING-ACL, pp. 483–490 (1998)
5. Koskenniemi, K.: A General Computational Model for Word-form Recognition and Production. In: 22nd Annual Meeting on Association for Computational Linguistics, pp. 178–181 (1984)
6. Lewis, G.: Turkish Grammar. Oxford University Press, Oxford (2001)
7. Megyesi, B.: Improving Brill's PoS Tagger for an Agglutinative Language. In: Joint Sigdat Conference on Empirical Methods in Natural Language Processing and Very Large Corpora (1999)
8. Mohri, M.: Finite-state Transducers in Language and Speech Processing. Computational Linguistics 23(2), 269–311 (1997)
9. Oflazer, K., Tür, G.: Morphological Disambiguation by Voting Constraints. In: ACL, pp. 222–229 (1997)
10. Sak, H., Güngör, T., Saraçlar, M.: Morphological Disambiguation of Turkish Text with Perceptron Algorithm. In: Gelbukh, A. (ed.) CICLing 2007. LNCS, vol. 4394, pp. 107–118. Springer, Heidelberg (2007)
11. Salor, Ö., Pellom, B.L., Çiloğlu, T., Hacıoğlu, K., Demirekler, M.: On Developing New Text and Audio Corpora and Speech Recognition Tools for the Turkish Language. In: ICSLP (2002)
12. Say, B., Zeyrek, D., Oflazer, K., Özge, U.: Development of a Corpus and a Treebank for Present-day Written Turkish. In: 11th International Conference of Turkish Linguistics (2002)
13. Yüret, D., Türe, F.: Learning Morphological Disambiguation Rules for Turkish. In: HLT-NAACL (2006)