Implementing Voting Constraints with Finite State Transducers

Kemal Oflazer and Gökhan Tür
Department of Computer Engineering and Information Science
Bilkent University, Bilkent, Ankara, TR-06533, Turkey
{ko,tur}@cs.bilkent.edu.tr

Abstract:

We describe a finite state transducer implementation of a constraint-based morphological disambiguation system in which individual constraint rule vote on matching morphological parses. Voting constraint rules have a number of desirable properties: The outcome of the disambiguation is *independent* of the order of application of the local contextual constraint rules. Thus the rule developer is relieved from worrying about conflicting rule sequencing. The approach can also combine statistically and manually obtained constraints, and incorporate negative constraints that rule out certain patterns. The transducer implementation has a number of desirable properties compared to other finite state tagging and light parsing approaches, implemented with automata intersection. The most important of these is that since constraints do not remove parses there is no risk of an overzealous constraint "killing a sentence" by removing all parses of a token during intersection. After a description of our approach we present preliminary results from tagging the Wall Street Journal Corpus with this approach. With about 400 statistically derived constraints and about 570 manual constraints, we can attain an accuracy of 97.82% on the training corpus and 97.29% on the test corpus. We then describe a finite state implementation of our approach and discuss various related issues.

1 Introduction

We describe a finite state implementation of a constraint-based morphological disambiguation system in which individual constraints vote on matching morphological parses and disambiguation of all tokens in a sentence is performed at the end, by selecting parses that collectively make up the the highest voted combination. The approach depends on assigning votes to constraints via statistical and/or manual means, and then letting constraint rule cast votes on matching parses of a given lexical item. This approach does not reflect the outcome of matching constraint rules to the set of morphological parses immediately. Only after all applicable rules are applied to a sentence, all tokens are disambiguated in parallel. Thus, the outcome of the rule applications is independent of the order of rule applications.

Constraint-based morphological disambiguation systems (e.g. [OK94, Vou95, Kos90]) typically look at a context of several sequential tokens each annotated with their possible morphological interpretations (or tags), and in a reductionistic way, remove parses that are considered to be impossible in the given context. Since constraint rule application is ordered, parses removed by one rule may not be used or referred to in subsequent rule applications. Addition of a new rule requires that its place in the sequence be carefully determined to avoid any unwanted interactions. Automata intersection based approaches run the risk of deleting all parses of a sentence, and also have been observed to end up with large intersected machines. Our approach eliminates the ordering problem as parse removals are not committed during application, but only after all rules are processed. Figure 1 highlights the voting constraints paradigm.

```
W1        W2        W3        W4        Wn   Tokens

t1        t1        t1        t1   ...   t1

          t2        t2        t2         t2   Parses/Tags

                    t3        t3         t3


R1        R3        R2        ...   Rm   Voting Rules
```
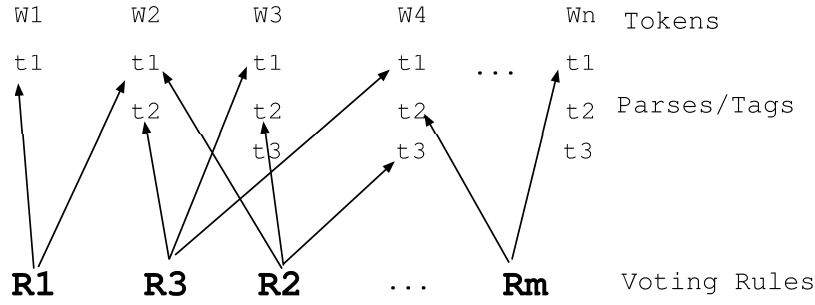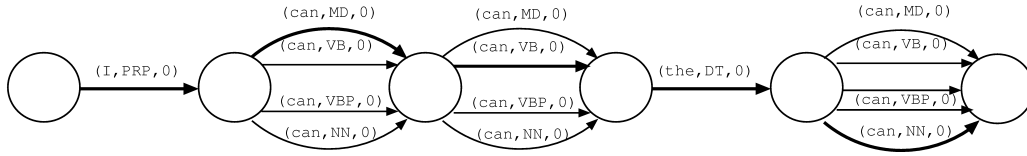
Figure 1: Voting Constraint Rules

Figure 2: Representing Sentences with a directed acyclic graph

In the following sections we describe voting constraint rules and then some present preliminary results from tagging English. We then present an implementation using finite state transducers and discuss various issues involved.

2   Voting Constraints

Voting constraints operate on sentences where each token has been assigned all possible tags by a lexicon or by a morphological analyzer. We represent a sentence consisting n tokens $w_1, w_2, \ldots w_n$, each with morphological parses/tags $t_{i,1}, t_{i,2}, \ldots, t_{i,a_i}$, a_i being the number of ambiguous parses for token i, as a standard chart using a directed acyclic graph (DAG). The nodes represent token boundaries and arcs are labeled with triplets of the sort $L = (w_i, t_{i,j}, v_{i,j})$ where $v_{i,j}$, (initially 0), is the total vote associated with tag $t_{i,j}$ of w_i. For instance, the sentence ``I can can the can.'' would initially be represented by the graph shown in Figure 2 where bold arcs denote the contextually correct tags.

We describe constraints on the ambiguous interpretation of tokens using rules with two components $R = (C_1, C_2, \cdots, C_n; V)$ where the C_i are, in general, feature constraints on a sequence of the ambiguous parses, and V is an integer denoting the vote of the rule. For English, the features that we use are TAG and LEX, but is certainly possibly to extend the set of features used, by including features such as initial letter capitalization, any derivational information, etc.

The following examples illustrate some rules:

1. ([TAG=MD],[TAG=VB]; 100) and ([TAG=MD],[TAG=RB],[TAG=VB]; 100) are two constraints with a high vote to encourage modal followed a verb possibly interrupted by an adverb.

2. ([TAG=DT,LEX=that],[TAG=NNS]; -100) discourages a singular determiner reading of that before a plural noun.

2

3. (`[TAG=DT,LEX=each]` `[TAG=JJ,LEX=other]`;100) is a rule with a high vote that captures a collocation [San95].

The constraints apply to a sentence in the following manner. Assume for a moment, all possible paths from the start node to the end node of a sentence DAG are explicitly enumerated. For each path, we apply each constraint to all possible sequences of token parses. For instance, let $R = (C_1, C_2, \cdots, C_m; V)$ be a constraint and let $L_i, L_{i+1}, \cdots, L_{i+m-1}$ be some sequence of labels labeling sequential arcs of a path. We say R matches this sequence of parses if tag and token components of $L_j, i \leq j \leq i + m - 1$ subsumed by the corresponding constraint C_{j-i+1}. When such a rule matches, the votes of parses matching are incremented by V. When all constraints are applied to all possible sequences in all paths, we select the path(s) with the maximum total tallied vote for the parses on it. If there are multiple paths with the same maximum vote, the tokens whose parses are different in these paths are assumed to be left ambiguous.

Given that in English each token has on the average about two tags, the procedural description above is very inefficient for all but very short sentences. A quite efficient procedure for implementing this operation based on Church windowing idea [Chu88] has been described by Tür and Oflazer [TO98]. Oflazer and Tür [OT97] presents an application of essentially the same approach (augmented with some additional statistical help) to morphological disambiguation of Turkish.

3   Preliminary Results from Tagging English

We have experimented with this approach using the Wall Street Journal Corpus from the Penn Treebank CD. We used two classes of constraints: one class derived from the training corpus (a set of 5000 sentences (about 109,000 tokens) from the WSJ Corpus) and a second set of hand-crafted constraints mainly incorporating negative constraints (demoting impossible or unlikely situations) or lexicalized positive constraints. These were constructed by observing the failures of the statistical constraints on the training corpus and repairing them. A test corpus of 500 sentences (about 11,500 tokens) was set aside for testing.

For the statistical constraints, we extracted tag k- grams from the tagged training corpus for $k = 2, 3, 4$, and 5. For each tag k-gram, we computed a vote which is essentially very similar to the weights used by Tzoukermann *et al.* [TRG95] except that we do not use their notion of genotypes exactly in the same way. Given a tag k-gram $t_1, t_2, \ldots t_k$, let

$$n = count(t_1 \in Tags(w_i), t_2 \in Tags(w_{i+1}), \ldots, t_k \in Tags(w_{i+k-1}))$$

for all possible i in the training corpus, be the number of possible places the tags sequence *can* possibly occur. Here $Tags(w_i)$ is the set of tags associated with token w_i. Let f be the number of times the tag sequence $t_1, t_2, \ldots t_k$ actually occurs in the tagged text, that is $f = count(t_1, t_2, \ldots t_k)$. We smooth f/n by defining $p = \frac{f+0.5}{n+1}$ so that neither p nor $1 - p$ is zero. The uncertainty of p is given by $\sqrt{p(1-p)/n}$ [TRG95]. We then compute the vote for this k-gram as

$$Vote(t_1, t_2, \ldots t_k) = (p - \sqrt{p(1-p)/n}) * 100.$$

This formulation thus gives high votes to k-grams which are selected most of the time they are "selectable." And, among the k-grams which are equally good (same f/n), those with a higher n (hence less uncertainty) are given higher votes. The votes for negative and positive hand-crafted

constraints are selected to override any vote the statistical constraints may have. The initial lexical votes for the parse $t_{i,j}$ of token w_i are obtained from the training corpus the usual way, i.e., as $count(w_i, t_{i,j})/count(w_i)$ normalized to between 0 and 100.

After extracting the k-grams as described above for $k = 2, 3, 4$ and 5, we ordered each group by decreasing votes and did an initial set of experiments with these, to select a small group of constraints that performed satisfactorily. Table 1 presents, for reference, the number of distinct k-grams extracted and how they performed when they solely were used as constraints. We selected after this experimen-

k	No. of k-grams	Train. Set Accuracy	Test Set Accuracy
2	867	97.78	95.70
3	8315	97.99	96.87
4	27871	98.88	96.56
5	54780	99.61	95.84

Table 1: Performance with 2,3, 4 and 5-gram voting constraints

tation, the first 200 (with highest votes) of the bi-gram and first 200 of the 3-gram constraints, as the set of statistical constraints; inclusion of 4- and 5-grams with highest votes did not have meaningful major impact on the results *It should be noted that the constraints obtained this way are purely constraints on tag sequences and do not use any lexical or genotype information.* The initial lexical votes were obtained from the training corpus also as described above.[1] We started tagging the training set with this set of constraint and by observing errors made and introducing hand-crafted rules arrived at a total of about 970 constraints. Most of the hand-crafted constraints were negative constraints (with large negative votes) to disallow certain tag sequences. Table 2 presents a set of tagging result from this experimentation. Although are results are quite preliminary, we feel that the results in the last

Constraint Set	Train. Set Accuracy	Test Set Accuracy
1	95.37	94.13
1+2	96.37	95.38
1+3	96.18	94.99
1+2+3	96.65	95.80
1+4	97.13	96.48
1+2+4	97.74	97.08
1+3+4	97.41	96.77
1+2+3+4	97.82	97.29

(1) Lexical Votes Only (2) 200 2-grams (3) 200 3-grams (4) 570 Manual Constraints

Table 2: Results from tagging with both statistically and manually derived voting constraints rules

row of Table 2 are quite satisfactory and warrant further extensive investigation.

---

[1]Thus the ambiguities of the tokens were limited to the ones found in the training corpus.

4 Implementing Voting Constraints with Finite State Transducers

The approach described above can be implemented by finite state transducers. For this, we view the parses of the tokens making up a sentence as making up a acyclic a finite state recognizer (or an identity transducer [KK94]). The states mark word boundaries. Transitions are labeled with labels are of the sort $L = (w_i, t_{i,j}, v_{i,j})$, and the rightmost node denotes the final state.

This approach is very different from that of Roche and Schabes [RS95] who use transducers to implement Brill's transformation-based tagging approach [Bri95]. It shares certain concepts with Tzoukermann and Radev's use of weighted finite state transducers for tagging [TR97] in that both approaches combine statistical and hand-crafted linguistic information but employ finite state devices in very different ways.

The basic idea behind using finite state transducers is that the voting constraint rules can be represented as transducers which increment the votes of the matching input sequence segments by an appropriate amount, but ignore and pass through unchanged, segments they are not sensitive to. When an identity finite state transducer corresponding to the input sentence is composed with a constraint transducer, the output is a slightly modified version of the sentence transducer with possibly additional transitions and states, where the votes of some of the transition arc labels have been appropriately incremented. When the sentence transducer is composed with all the constraint transducers in sequence, all possible votes are cast and the final sentence transducer reflects all the votes. The parses on the path with the highest total vote from the start to one of the final states, can then be selected. *The key point here is that due to the nature of the composition operator, the constraint transducers can, if necessary, be composed off-line first, giving a single constraint transducer and then this one is composed with every sentence transducer once.*

Using a finite state framework provides, trivially, some additional descriptive advantages in describing rules. For instance, one can use rules involving the Kleene star so that single rule such as (`[TAG=MD]`, `[TAG=RB]*,[TAG=VB]; 100`) can deal with any number of intervening adverbials.[2]

4.1 The Transducer Architecture

We use the Xerox Finite State Tools to implement our approach. The finite state transducer system consists of the following components depicted in of Figure 3.

4.1.1 The lexicon transducer

The lexicon transducer implements `[ L [" "]+ ]*`[3] where the transducer `L` maps a token to all its possible tags/parses, at the same time inserting the relevant lexical votes for each parse. In our current implementation for English, the transducer `L` is the union of a set of transducers of the sort

---

[2]Note that in this case the vote will be added to all the matching parses, thus depending on how many sequential parses match the *'ed constraint, the total vote contribution of the rules will differ. This may actually be desirable.

[3]We use the Xerox regular expression language (see http://www.xrce.xerox.com/research/mltt/fst/home.html) to describe our regular expressions.
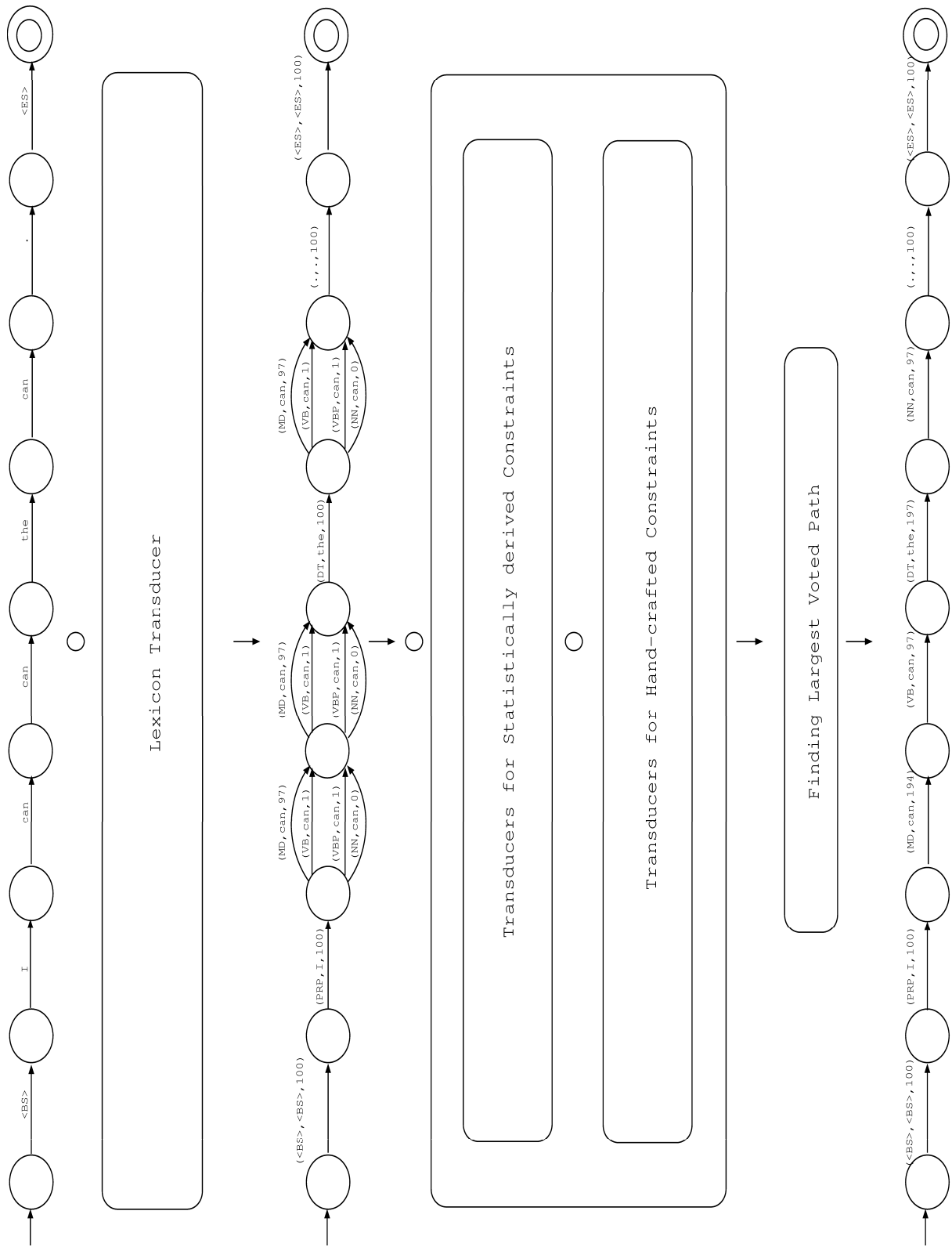
Figure 3: The Architecture of Voting Constraint Transducers

```
[s a i d] .x. [ "(" "VBD/"   s a i d      "<"  "+" 9 8 ">" ")" ] ]
[s a i d] .x. [ "(" "VBN/"   s a i d      "<"  "+"   1 ">" ")" ] ]
[s a i d] .x. [ "(" "JJ/"    s a i d      "<"  "+"   1 ">" ")" ] ]
```

so a "lookdown" of the token `said` will result at the lower side of the transducer outputs `(VBD/said<+98>)` `(VBN/said<+1>)` `(JJ/said<+1>)`. Thus when a sentence transducer (representing just the lexical items) is composed with the lexicon transducer as depicted at the top of Figure 3 one gets a transducers with lexical ambiguities and votes inserted, which can then be composed with the constraint transducers.

4.1.2 Voting Constraint Transducers

Each voting constraint is represented by a transducer that checks if the constraints imposed are satisfied on the input, and if so, increments the votes at the relevant input positions appropriately. In order to describe the transducer architecture more clearly, let us concentrate on a specific example rule:

    ([TAG=MD],[TAG=VB]; 100)

Let us assume that input is represented as a sequence of triplets of the sort `(tag word vote)`[4] The transducer corresponding to the regular expression below will increment the vote fields of a sequence of any two triplets by 100 provided the first one has tag `MD` and the second one has tag `VB`

```
[ "("  TAGS WORD VOTES ")"]*                                              (1)
 .o.
[ "("  "MD/" WORD VOTES ")" "(" "VB/" WORD  VOTES ")"  @-> "{" ... "}" ] (2)
 .o.
[                                                                         (3)
    [         "("  TAGS WORD VOTES  ")" ]  |

    [ "{" [  "(" TAGS  WORD  [ ADD100  ]  ")"
             "(" TAGS  WORD  [ ADD100  ]  ")"
         ]
      "}"
    ]
]*
 .o.
{ "{" -> [], "}" -> []};                                                  (4)
```

This transducer is the composition of four transducers (separated the composition operator `.o.`). The top transducer (1) constrains the input to valid triples.[5] The second transducer brackets any sequence of such triplets matching the given rule constraints with { and } using longest match bracket operator

---

[4] Please note that this is a slightly different order that described earlier. In practice, this order was found to generate smaller transducers during compositions.

[5] Here `WORD` denotes a regular expression which describes an arbitrary sequence of English characters, `TAGS` denotes a regular expression which is the union of all (possibly multi-character) tag symbols, and `VOTES` denotes a regular expression of the sort `"<" ['+"|"-"] DIGITS+ ">"` with DIGITS being the union of decimal digit symbols.

[Kar96].[6] Thus all segments in the input sequence where the constraints are all satisfied, are bracketed by this transducer. The third transducer (3) either passes through the unbracketed sections of the input (as indicated by the first part of the disjunct) or increments by 100 the vote fields of the triplets within the brackets { and }. The ADD100 is a transducer that "adds" 100 to the vote field of the triplet. It is the 99-fold composition of an ADD1 transducer with itself. The ADD1 transducer will add one to a (signed) number at its upper side input.[7] When compiled this constraint rule becomes a transducer with 75 states and 1,197 arcs.

Thus when all the transducers above are compiled, the compositions are computed at compile time and one gets a single transducer for the voting constraints. The transducers for all constraints are obtained in a similar way and composed off-line giving one big transducer which does the appropriate vote updates in appropriate places. In practice the final voting constraint transducer may be big, so instead, one can leave it as a cascade of a small number of transducers.

4.2   Operational Aspects

A sentence such as "I can can the can." is represented as the transducer corresponding to the the the regular expression

```
    [ <BS> I can can the can .   <ES>]
```
[8]

When this transducer is composed with the lexicon transducer, the resulting transducer corresponds to the following regular expression:

```
  [(<BS>/<BS><+100>)]
  [(PRP/I<+100>) ]
  [(MD/can<+97>) | (VB/can<+1>) | (NN/can<+1>) |  (VBP/can<+1>) ]
  [(MD/can<+97>) | (VB/can<+1>) | (NN/can<+1>) |  (VBP/can<+1>) ]
  [(DT/the<+100>) ]
  [(MD/can<+97>) | (VB/can<+1>) | (NN/can<+1>) |  (VBP/can<+1>) ]
  [(./.<+100>)]
  [(<ES>/<ES><+100>)]
```

which allows for 64 possible "readings". After this transducer is composed with the voting constraint transducer(s), one gets a transducer which still has 64 readings, but now the labels reflect votes from any matching constraints. A simple DAG longest path algorithm (e.g. [CLR91]) on the DAG of the resulting transducer gives the largest voted path as

```
(<BS>/<BS><+100>)
(PRP/I<+100>)(MD/can<+194>)(VB/can<+98>)(DT/the<+197>)(NN/can<97>)(./.<+100>)
(<ES>/<ES><+100>)
```

---

[6]Note that this simple version does not deal with rules whose constraints may overlap (e.g. ([TAG=NN],[TAG=NN];100)).

[7]This is a bit modified version of the transducer described at http://www.rxrc.xerox.com/research/mltt/fst/-fsexamples.html, that can deal with signed numbers. The ADD1 transducer can be composed with itself off line any number of times to get a transducer to add any number.

[8]For better readability, the spaces obligatory space between word symbols will not be shown from now on.

5   Implementation

We have developed two PERL-based rule compilers for compiling lexicon files and constraints into scripts which are then compiled into transducers by the Xerox finite state tools. In this section we provide some information about the transducers obtained from the WSJ Corpus experiments.

The lexicon transducer compiled from about 16,000 unique lexical tokens from the training set had 37,208 states, and 52,912 arcs. The three sets of constraints for 2-grams 3-grams and hand-crafted constraints (sets 2, 3 and 4 in Figure 2) were compiled separately into three constraint transducers of 19,954 states and 296,545 arcs, 56,910 states and 685,365 arcs and 334,215 states, 2,651,550 arcs, respectively. It is certainly possible to combine these transducers by composition at compile time. If size becomes a problem, one can have smaller transducers which are sequentially composed with the sentence transducer at tag time. For instance, when the hand-crafted constraints are split into 3 groups of about 200 each, the three resulting transducers are of size 63,865 states, 467,966 arcs, 44,831 states, 306,257 and 33,862 states, 233,401 arcs respectively, the collective size of which is less than the size of fully composed one. We have not really optimized the hand-crafted constraints for finite state compilation but it is certainly possible to reduce the number of such constraints by utilizing operators such as the Kleene star, complementation, etc.

Another observation during constraint compilation is that as constraints are being compiled, the size of intermediate compositions do not grow explosively. Thus the problem alluded to by Tapanainen in a similar approach [Tap97], does not seem to occur, since an intersection is not being computed. The results that we have provided earlier are from a C implementation. The tagging speed with the finite state transducers in the current environment is not very high since for each sentence the transducers have to be loaded from the disk. But with a suitable application interface to the lower level functions of the Xerox finite state environment, the tagging speed can be improved significantly.

The system deals with unknown words in a rather trivial way by attaching any meaningful open class word tags to unknown words and later picking the one(s) selected by the voting process.

6   Conclusions

We have presented an approach to constraint-based tagging and an implementation of the approach based on finite state transducers. The approach can combine both statistically and manually derived constraints and relieves the developer from worrying about conflicting rule application sequencing. Preliminary results from tagging the Wall Street Journal Corpus are quite promising. We would like to further evaluate our approach using 10-fold cross validation on the WSJ corpus and late on the Brown Corpus. We also would like to utilize the full expressive power of the regular expression operations to compact our constraint rule base and implement the transducer version directly rather than loading transducer files from the disk every time.

7 Acknowledgments

References

[Bri95]   Eric Brill. Transformation-based error-driven learning and natural language processing: A case study in part-of-speech tagging. *Computational Linguistics*, 21(4):543–566, December 1995.

[Chu88]   Kenneth W. Church. A stochastic parts program and a noun phrase parser for unrestricted text. In *Proceedings of the Second Conference on Applied Natural Language Processing*, Austin, Texas, 1988.

[CLR91]   Thomas H Cormen, Charles E. Leiserson, and Ronald L. Rivest. *Introduction to Algorithms*. The MIT Press and McGraw Hill, 1991.

[Kar96]   Lauri Karttunen. Directed replacement. In *Proceedings of the 34th Annual Meeting of the Association for Computational Linguistics*, pages 108–115, 1996.

[KK94]   Ronald M. Kaplan and Martin Kay. Regular models of phonological rule systems. *Computational Linguistics*, 20(3):331–378, September 1994.

[Kos90]   Kimmo Koskenniemi. Finite-state parsing and disambiguation. In *Proceedings of COLING-90*, volume 2, pages 229–232, 1990.

[OK94]   Kemal Oflazer and İlker Kuruöz. Tagging and morphological disambiguation of Turkish text. In *Proceedings of the 4th Applied Natural Language Processing Conference*, pages 144–149. ACL, October 1994.

[OT97]   Kemal Oflazer and Gökhan Tür. Morphological disambiguation by voting constraints. In *Proceedings of ACL'97/EACL'97, The 35th Annual Meeting of the Association for Computational Linguistics*, 1997.

[RS95]   Emmanuel Roche and Yves Schabes. Deterministic part-of-speech tagging with finite-state transducers. *Computational Linguistics*, 21(2):227–253, June 1995.

[San95]   Beatrice Santorini. Part-of-speech tagging guidelines. Available from http://www.ldc.upenn.edu, 1995.

[Tap97]   Pasi Tapanainen. Applying a finite-state intersection grammar. In Emmanuel Roche and Yves Schabes, editors, *Finite State Language Processing*, chapter 10. The MIT Press, 1997.

[TO98]   Gökhan Tür and Kemal Oflazer. Tagging English by path voting constraints. Submitted to COLING/ACL'98, 1998.

[TR97]   Evelyne Tzoukerman and Dragomir R. Radev. Use of weighted finite state transducers in part of speech tagging. Available from `http://xxx.lanl.gov/ps/cmp-lg/9710001`, 1997.

[TRG95] Evelyne Tzoukermann, Dragomir R. Radev, and William A. Gale. Combining linguistic knowledge and statistical learning in french part-of-speech tagging. In *Proceedings of the ACL SIGDAT Workshop* From Texts to Tags: Issues in Multilingual Language Analysis, pages 51–57, 1995.

[Vou95] Atro Voutilainen. Morphological disambiguation. In Fred Karlsson, Atro Voutilainen, Juha Heikkilä, and Arto Anttila, editors, *Constraint Grammar-A Language–Independent System for Parsing Unrestricted Text*, chapter 5. Mouton de Gruyter, 1995.