

Fast, Greedy Model Minimization for Unsupervised Tagging

Sujith Ravi and Ashish Vaswani and Kevin Knight and David Chiang

University of Southern California

Information Sciences Institute

{sravi, avaswani, knight, chiang}@isi.edu

Abstract

Model minimization has been shown to work well for the task of unsupervised part-of-speech tagging with a dictionary. In (Ravi and Knight, 2009), the authors invoke an integer programming (IP) solver to do model minimization. However, solving this problem exactly using an integer programming formulation is intractable for practical purposes. We propose a novel two-stage greedy approximation scheme to replace the IP. Our method runs fast, while yielding highly accurate tagging results. We also compare our method against standard EM training, and show that we consistently obtain better tagging accuracies on test data of varying sizes for English and Italian.

1 Introduction

The task of unsupervised part-of-speech (POS) tagging with a dictionary as formulated by Meri- aldo (1994) is: given a raw word sequence and a dictionary of legal POS tags for each word type, tag each word token in the text. A common approach to modeling such sequence labeling problems is to build a bigram Hidden Markov Model (HMM) parameterized by *tag-bigram* transition probabilities $P(t_i|t_{i-1})$ and *word-tag* emission probabilities $P(w_i|t_i)$. Given a word sequence w and a tag sequence t , of length N , the joint probability $P(w, t)$ is given by:

$$P(w, t) = \prod_{i=1}^N P(w_i|t_i) \cdot P(t_i|t_{i-1}) \quad (1)$$

We can train this model using the Expectation Maximization (EM) algorithm (Dempster and Rubin, 1977) which learns $P(w_i|t_i)$ and $P(t_i|t_{i-1})$ that maximize the likelihood of the observed data. Once the parameters are learnt, we can find the best tagging using the Viterbi algorithm.

$$\hat{t} = \arg \max_t P(w, t) \quad (2)$$

Ravi and Knight (2009) attack the Meri- aldo task in two stages. In the first stage, they search for a minimized transition model (i.e., the smallest set of tag bigrams) that can explain the data using an integer programming (IP) formulation. In the second stage, they build a smaller HMM by restricting the transition parameters to only those tag bigrams selected in the minimization step. They employ the EM algorithm to train this model, which prunes away some of the emission parameters. Next, they use the pruned emission model along with the original transition model (which uses the full set of tag bigrams) and re-train using EM. This alternating EM training procedure is repeated until the number of tag bigrams in the Viterbi tagging output does not change between subsequent iterations. The final Viterbi tagging output from their method achieves state-of-the-art accuracy for this task. However, their minimization step involves solving an integer program, which can be very slow, especially when scaling to large-scale data and more complex tagging problems which use bigger tagsets. In this paper, we present a novel method that optimizes the same objective function using a fast greedy model selection strategy. Our contributions are summarized below:

- We present an efficient two-phase greedy-selection method for solving the minimization objective from Ravi and Knight (2009), which runs much faster than their IP.
- Our method easily scales to large data sizes (and big tagsets), unlike the previous minimization-based approaches and we show runtime comparisons for different data sizes.
- We achieve very high tagging accuracies comparable to state-of-the-art results for unsupervised POS tagging for English.
- Unlike previous approaches, we also show results obtained when testing on the entire Penn Treebank data (973k word tokens) in addition to the standard 24k test data used for this task. We also show the effectiveness of this approach for Italian POS tagging.

2 Previous work

There has been much work on the unsupervised part-of-speech tagging problem. Goldwater and Griffiths (2007) also learn small models employing a fully Bayesian approach with sparse priors. They report 86.8% tagging accuracy with manual hyperparameter selection. Smith and Eisner (2005) design a *contrastive estimation* technique which yields a higher accuracy of 88.6%. Goldberg et al. (2008) use linguistic knowledge to initialize the parameters of the HMM model prior to EM training. They achieve 91.4% accuracy. Ravi and Knight (2009) use a Minimum Description Length (MDL) method and achieve the best results on this task thus far (91.6% word token accuracy, 91.8% with random restarts for EM). Our work follows a similar approach using a model minimization component and alternate EM training.

Recently, the integer programming framework has been widely adopted by researchers to solve other NLP tasks besides POS tagging such as semantic role labeling (Punyakanok et al., 2004), sentence compression (Clarke and Lapata, 2008), decipherment (Ravi and Knight, 2008) and dependency parsing (Martins et al., 2009).

3 Model minimization formulated as a Path Problem

The complexity of the model minimization step in (Ravi and Knight, 2009) and its proposed approximate solution can be best understood if we formulate it as a path problem in a graph.

Let $w = w_0, w_1, \dots, w_N, w_{N+1}$ be a word sequence where w_1, \dots, w_N are the input word tokens and $\{w_0, w_{N+1}\}$ are the *start/end* tokens. Let $T = \{T_1, \dots, T_K\} \cup \{T_0, T_{K+1}\}$ be the fixed set of all possible tags. T_0 and T_{K+1} are special tags that we add for convenience. These would be the *start* and *end* tags that one typically adds to the HMM lattice. The tag dictionary D contains entries of the form (w_i, T_j) for all the possible tags T_j that word token w_i can have. We add entries (w_0, T_0) and (w_{K+1}, T_{K+1}) to D . Given this input, we now create a directed graph $G(V, E)$. Let C_0, C_1, \dots, C_{K+1} be columns of nodes in G , where column C_i corresponds to word token w_i . For all $i = 0, \dots, N+1$ and $j = 0, \dots, K+1$, we add node $C_{i,j}$ in column C_i if $(w_i, T_j) \in D$. Now, $\forall i = 0, \dots, N$, we create directed edges from every node in C_i to every node in C_{i+1} . Each of these edges $e = (C_{i,j}, C_{i+1,k})$ is given the label (T_j, T_k) which corresponds to a tag bigram. This creates our directed graph. Let $l(e)$ be the tag bigram label of edges $e \in E$. For every path P from $C_{0,0}$ to $C_{N+1,K+1}$, we say that P uses an edge label or tag bigram (T_j, T_k) if there exists an edge e in P such that $l(e) = (T_j, T_k)$. We can now formulate the optimization problem as: *Find the smallest set S of tag bigrams such that there exists at least one path from $C_{0,0}$ to $C_{N+1,K+1}$ using only the tag bigrams in S .* Let us call this the Minimal Tag Bigram Path (MinTagPath) problem.

Figure 1 shows an example graph where the input word sequence is w_1, \dots, w_4 and $T = \{T_1, \dots, T_3\}$ is the input tagset. We add the start/end word tokens $\{w_0, w_5\}$ and corresponding tags $\{T_0, T_4\}$. The edges in the graph are instantiated according to the word/tag dictionary D provided as input. The node and edge labels are also illustrated in the graph. Our goal is to find a path from $C_{0,0}$ to $C_{5,4}$ using the smallest set of tag bigrams.

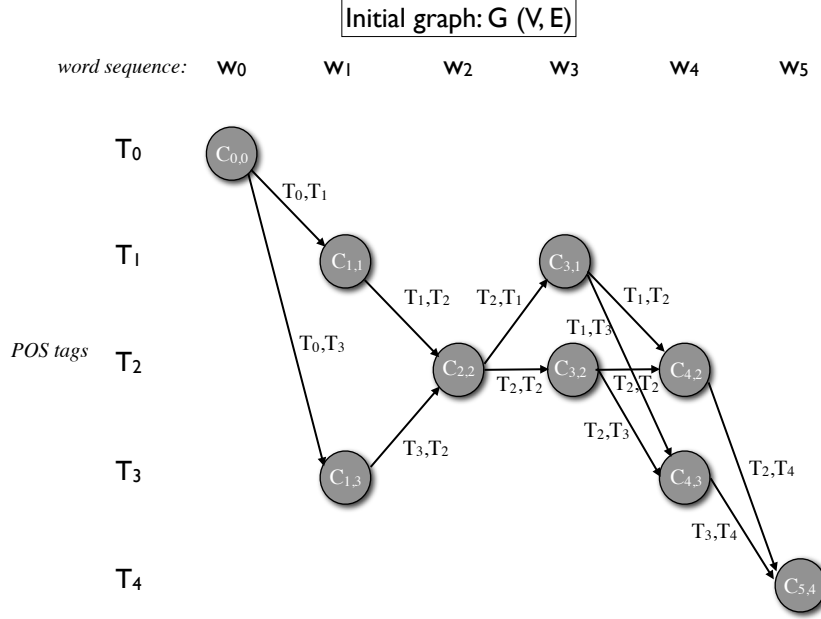


Figure 1: Graph instantiation for the MinTagPath problem.

4 Problem complexity

Having defined the problem, we now show that it can be solved in polynomial time even though the number of paths from $C_{0,0}$ to $C_{N+1,K+1}$ is exponential in N , the input size. This relies on the assumption that the tagset T is fixed in advance, which is the case for most tagging tasks.¹ Let B be the set of all the tag bigram labels in the graph, $B = \{l(e), \forall e \in E\}$. Now, the size of B would be at most $K^2 + 2K$ where every word could be tagged with every possible tag. For $m = 1 \dots |B|$, let B^m be the set of subsets of B each of which have size m . Algorithm 1 optimally solves the MinTagPath problem.

Algorithm 1 basically enumerates all the possible subsets of B , from the smallest to the largest, and checks if there is a path. It exits the first time a path is found and therefore finds the smallest possible set s_i of size m such that a path exists that uses only the tag bigrams in s_i . This implies the correctness of the algorithm. To check for path existence, we could either throw away all the edges from E not having a label in s_i , and then execute a Breadth-First-Search (BFS) or we could traverse

¹If K , the size of the tagset, is a variable as well, then we suspect the problem is NP-hard.

Algorithm 1 Brute Force solution to MinTagPath

```

for  $m = 1$  to  $|B|$  do
  for  $s_i \in B^m$  do
    Use Breadth First Search (BFS) to check
    if  $\exists$  path  $P$  from  $C_{0,0}$  to  $C_{N+1,K+1}$  using
    only the tag bigrams in  $s_i$ .
    if  $P$  exists then
      return  $s_i, m$ 
    end if
  end for
end for

```

only the edges with labels in s_i during BFS. The running time of Algorithm 1 is easy to calculate. Since, in the worst case we go over all the subsets of size $m = 1, \dots, |B|$ of B , the number of iterations we can perform is at most $2^{|B|}$, the size of the powerset \mathcal{P} of B . In each iteration, we do a BFS through the lattice, which has $O(N)$ time complexity² since the lattice size is linear in N and BFS is linear in the lattice size. Hence the running time is $\leq 2^{|B|} \cdot O(N) = O(N)$. Even though this shows that MinTagPath can be solved in polynomial time, the time complexity is prohibitively large. For the Penn Treebank, $K = 45$ and the

²Including throwing away edges or not.

worst case running time would be $\approx 10^{13.55} \cdot N$. Clearly, for all practical purposes, this approach is intractable.

5 Greedy Model Minimization

We do not know of an efficient, exact algorithm to solve the MinTagPath problem. Therefore, we present a simple and fast two-stage greedy approximation scheme. Notice that an optimal path P (or any path) *covers* all the input words i.e., every word token w_i has one of its possible taggings in P . Exploiting this property, in the first phase, we set our goal to cover all the word tokens using the least possible number of tag bigrams. This can be cast as a set cover problem (Garey and Johnson, 1979) and we use the set cover greedy approximation algorithm in this stage. The output tag bigrams from this phase might still not allow any path from $C_{0,0}$ to $C_{N+1,K+1}$. So we carry out a second phase, where we greedily add a few tag bigrams until a path is created.

5.1 Phase 1: Greedy Set Cover

In this phase, our goal is to cover all the word tokens using the least number of tag bigrams. The covering problem is exactly that of set cover. Let $U = \{w_0, \dots, w_{N+1}\}$ be the set of elements that needs to be covered (in this case, the word tokens). For each tag bigram $(T_i, T_j) \in B$, we define its corresponding covering set S_{T_i, T_j} as follows:

$$S_{T_i, T_j} = \left\{ w_n : \begin{aligned} &((w_n, T_i) \in D \\ &\wedge (C_{n,i}, C_{n+1,j}) \in E \\ &\wedge l(C_{n,i}, C_{n+1,j}) = (T_i, T_j)) \\ &\vee ((w_n, T_j) \in D \\ &\wedge (C_{n-1,i}, C_{n,j}) \in E \\ &\wedge l(C_{n-1,i}, C_{n,j}) = (T_i, T_j)) \end{aligned} \right\}$$

Let the set of covering sets be X . We assign a cost of 1 to each covering set in X . The goal is to select a set $CHOSEN \subseteq X$ such that $\bigcup_{S_{T_i, T_j} \in CHOSEN} S_{T_i, T_j} = U$, minimizing the total cost of $CHOSEN$. This corresponds to covering all the words with the least possible number of tag bigrams. We now use the greedy approximation algorithm for set cover to solve this problem. The pseudo code is shown in Algorithm 2.

Algorithm 2 Set Cover : Phase 1

Definitions

Define $CAND$: Set of candidate covering sets in the current iteration

Define U_{rem} : Number of elements in U remaining to be covered

Define $E_{S_{T_i, T_j}}$: Current effective cost of a set

Define Itr : Iteration number

Initializations

LET $CAND = X$

LET $CHOSEN = \emptyset$

LET $U_{rem} = U$

LET $Itr = 0$

LET $E_{S_{T_i, T_j}} = \frac{1}{|S_{T_i, T_j}|}, \forall S_{T_i, T_j} \in CAND$

while $U_{rem} \neq \emptyset$ **do**

$Itr \leftarrow Itr + 1$

Define $\hat{S}_{Itr} = \underset{S_{T_i, T_j} \in CAND}{\operatorname{argmin}} E_{S_{T_i, T_j}}$

$CHOSEN = CHOSEN \cup \hat{S}_{Itr}$

Remove \hat{S}_{Itr} from $CAND$

Remove all the current elements in \hat{S}_{Itr} from U_{rem}

Remove all the current elements in \hat{S}_{Itr} from every $S_{T_i, T_j} \in CAND$

Update effective costs, $\forall S_{T_i, T_j} \in CAND$, $E_{S_{T_i, T_j}} = \frac{1}{|S_{T_i, T_j}|}$

end while

return $CHOSEN$

For the graph shown in Figure 1, here are a few possible covering sets S_{T_i, T_j} and their initial effective costs $E_{S_{T_i, T_j}}$.

- $S_{T_0, T_1} = \{w_0, w_1\}, E_{S_{T_0, T_1}} = 1/2$
- $S_{T_1, T_2} = \{w_1, w_2, w_3, w_4\}, E_{S_{T_1, T_2}} = 1/4$
- $S_{T_2, T_2} = \{w_2, w_3, w_4\}, E_{S_{T_2, T_2}} = 1/3$

In every iteration Itr of Algorithm 2, we pick a set \hat{S}_{Itr} that is most cost effective. The elements that \hat{S}_{Itr} covers are then removed from all the remaining candidate sets and U_{rem} and the effectiveness of the candidate sets is recalculated for the next iteration. The algorithm stops when all elements of U i.e., all the word tokens are covered. Let, $B_{CHOSEN} = \{(T_i, T_j) : S_{T_i, T_j} \in$

$CHOSEN\}$, be the set of tag bigrams that have been chosen by set cover. Now, we check, using BFS, if there exists a path from $C_{0,0}$ to $C_{N+1,K+1}$ using only the tag bigrams in B_{CHOSEN} . If not, then we have to add tag bigrams to B_{CHOSEN} to enable a path. To accomplish this, we carry out the second phase of this scheme with another greedy strategy (described in the next section).

For the example graph in Figure 1, one possible solution $B_{CHOSEN} = \{(T_0, T_1), (T_1, T_2), (T_2, T_4)\}$.

5.2 Phase 2: Greedy Path Completion

We define a graph $G_{CHOSEN}(V', E') \subseteq G(V, E)$ that contains the edges $e \in E$ such $l(e) \in B_{CHOSEN}$.

Let $B_{CAND} = B \setminus B_{CHOSEN}$, be the current set of candidate tag bigrams that can be added to the final solution which would create a path. We would like to know how many *holes* a particular tag bigram (T_i, T_j) can fill. We define a hole as an edge e such that $e \in G \setminus G_{CHOSEN}$ and there exists $e', e'' \in G_{CHOSEN}$ such that $tail(e') = head(e) \wedge tail(e) = head(e'')$.

Figure 2 illustrates the graph G_{CHOSEN} using tag bigrams from the example solution to Phase 1 (Section 5.1). The dotted edge $(C_{2,2}, C_{3,1})$ represents a hole, which has to be filled in the current phase in order to complete a path from $C_{0,0}$ to $C_{5,4}$.

In Algorithm 3, we define the effectiveness of a candidate tag bigram $H(T_i, T_j)$ to be the number of holes it covers. In every iteration, we pick the most effective tag bigram, fill the holes and recalculate the effectiveness of the remaining candidate tag bigrams.

Algorithm 3 returns B_{FINAL} , the final set of chosen tag bigrams. It terminates when a path has been found.

5.3 Fitting the Model

Once the greedy algorithm terminates and returns a minimized grammar of tag bigrams, we follow the approach of Ravi and Knight (2009) and fit the minimized model to the data using the alternating EM strategy. The alternating EM iterations are terminated when the change in the size of the observed grammar (i.e., the number of unique tag

Algorithm 3 Greedy Path Complete : Phase 2

Define B_{FINAL} : Final set of tag bigrams selected by the two-phase greedy approach

LET $B_{FINAL} = B_{CHOSEN}$

LET $H(T_i, T_j) = |\{e\}|$ such that $l(e) = (T_i, T_j)$ and e is a hole, $\forall (T_i, T_j) \in B_{CAND}$

while \nexists path P from $C_{0,0}$ to $C_{N+1,K+1}$ using only $(T_i, T_j) \in B_{CHOSEN}$ **do**

Define $(\hat{T}_i, \hat{T}_j) = \operatorname{argmax}_{(T_i, T_j) \in B_{CAND}} H(T_i, T_j)$

$B_{FINAL} = B_{FINAL} \cup (\hat{T}_i, \hat{T}_j)$

Remove (\hat{T}_i, \hat{T}_j) from B_{CAND}

$G_{CHOSEN} = G_{CHOSEN} \cup \{e\}$ such that $l(e) = (T_i, T_j)$

$\forall (T_i, T_j) \in B_{CAND}$, Recalculate $H(T_i, T_j)$

end while

return B_{FINAL}

bigrams in the tagging output) is $\leq 5\%$. We refer to our entire approach using greedy minimization followed by EM training as MIN-GREEDY.

6 Experiments and Results

6.1 English POS Tagging

Data: We use a standard test set (consisting of 24,115 word tokens from the Penn Treebank) for the POS tagging task (described in Section 1). The tagset consists of 45 distinct tag labels and the dictionary contains 57,388 word/tag pairs derived from the entire Penn Treebank. Per-token ambiguity for the test data is about 1.5 tags/token. In addition to the standard 24k dataset, we also train and test on larger data sets of 48k, 96k, 193k, and the entire Penn Treebank (973k).

Methods: We perform comparative evaluations for POS tagging using three different methods:

1. **EM:** Training a bigram HMM model using EM algorithm.
2. **IP:** Minimizing grammar size using integer programming, followed by EM training (Ravi and Knight, 2009).
3. **MIN-GREEDY:** Minimizing grammar size using the Greedy method described in Sec-

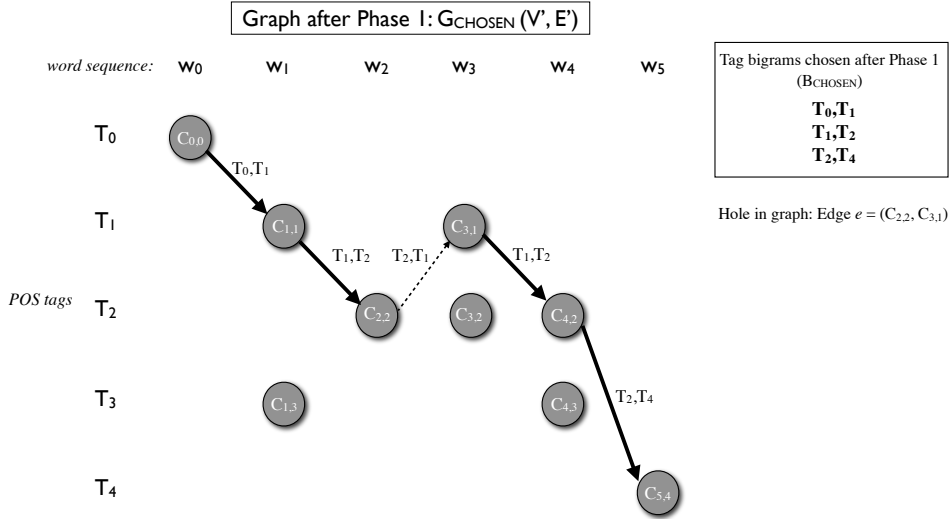


Figure 2: Graph constructed with tag bigrams chosen in Phase 1 of the MIN-GREEDY method.

tion 5, followed by EM training.

Results: Figure 3 shows the tagging performance (word token accuracy %) achieved by the three methods on the standard test (24k tokens) as well as Penn Treebank test (PTB = 973k tokens). On the 24k test data, the MIN-GREEDY method achieves a high tagging accuracy comparable to the previous best from the IP method. However, the IP method does not scale well which makes it infeasible to run this method in a much larger data setting (the entire Penn Treebank). MIN-GREEDY on the other hand, faces no such problem and in fact it achieves high tagging accuracies on all four datasets, consistently beating EM by significant margins. When tagging all the 973k word tokens in the Penn Treebank data, it produces an accuracy of 87.1% which is much better than EM (82.3%) run on the same data.

Ravi and Knight (2009) mention that it is possible to interrupt the IP solver and obtain a sub-optimal solution faster. However, the IP solver did not return any solution when provided the same amount of time as taken by MIN-GREEDY for any of the data settings. Also, our algorithms were implemented in Python while the IP method employs the best available commercial software package (CPLEX) for solving integer programs.

Figure 4 compares the running time efficiency for the IP method versus MIN-GREEDY method

Test set	Efficiency (average running time in secs.)	
	IP	MIN-GREEDY
24k test	93.0	34.3
48k test	111.7	64.3
96k test	397.8	93.3
193k test	2347.0	331.0
PTB (973k) test	*	1485.0

Figure 4: Comparison of MIN-GREEDY versus MIN-GREEDY approach in terms of *efficiency* (average running time in seconds) for different data sizes. All the experiments were run on a single machine with a 64-bit, 2.4 GHz AMD Opteron 850 processor.

as we scale to larger datasets. Since the IP solver shows variations in running times for different datasets of the same size, we show the average running times for both methods (for each row in the figure, we run a particular method on three different datasets with similar sizes and average the running times). The figure shows that the greedy approach can scale comfortably to large data sizes, and a complete run on the entire Penn Treebank data finishes in just 1485 seconds. In contrast, the IP method does not scale well—on average, it takes 93 seconds to finish on the 24k test (versus 34 seconds for MIN-GREEDY) and on the larger PTB test data, the IP solver runs for

Method	Tagging accuracy (%)				
	when training & testing on:				
	24k	48k	96k	193k	PTB (973k)
EM	81.7	81.4	82.8	82.0	82.3
IP	91.6	89.3	89.5	91.6	*
MIN-GREEDY	91.6	88.9	89.4	89.1	87.1

Figure 3: Comparison of tagging accuracies on test data of varying sizes for the task of unsupervised English POS tagging with a dictionary using a 45-tagset. (* IP method does not scale to large data).

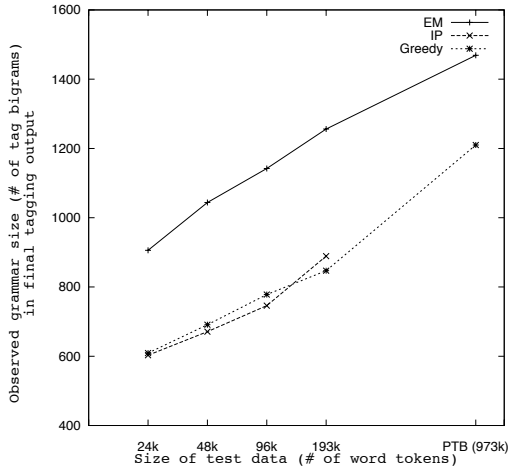


Figure 5: Comparison of observed grammar size (# of tag bigram types) in the final tagging output from EM, IP and MIN-GREEDY.

more than 3 hours without returning a solution.

It is interesting to see that for the 24k dataset, the greedy strategy finds a grammar set (containing only 478 tag bigrams). We observe that MIN-GREEDY produces 452 tag bigrams in the first minimization step (phase 1), and phase 2 adds another 26 entries, yielding a total of 478 tag bigrams in the final minimized grammar set. That is almost as good as the optimal solution (459 tag bigrams from IP) for the same problem. But MIN-GREEDY clearly has an advantage since it runs much faster than IP (as shown in Figure 4). Figure 5 shows a plot with the size of the observed grammar (i.e., number of tag bigram types in the final tagging output) versus the size of the test data for EM, IP and MIN-GREEDY methods. The figure shows that unlike EM, the other two approaches reduce the grammar size considerably and we observe the same trend even when scaling

Test set	Average Speedup	Optimality Ratio
24k test	2.7	0.96
48k test	1.7	0.98
96k test	4.3	0.98
193k test	7.1	0.93

Figure 6: Average speedup versus Optimality ratio computed for the model minimization step (when using MIN-GREEDY over IP) on different datasets.

to larger data. Minimizing the grammar size helps remove many spurious tag combinations from the grammar set, thereby yielding huge improvements in tagging accuracy over the EM method (Figure 3). We observe that for the 193k dataset, the final observed grammar size is greater for IP than MIN-GREEDY. This is because the alternating EM steps following the model minimization step add more tag bigrams to the grammar.

We compute the optimality ratio of the MIN-GREEDY approach with respect to the grammar size as follows:

$$\text{Optimality ratio} = \frac{\text{Size of IP grammar}}{\text{Size of MIN-GREEDY grammar}}$$

A value of 1 for this ratio implies that the solution found by MIN-GREEDY algorithm is exact. Figure 6 compares the optimality ratio versus average speedup (in terms of running time) achieved in the minimization step for the two approaches. The figure illustrates that our solution is nearly optimal for all data settings with significant speedup.

The MIN-GREEDY algorithm presented here can also be applied to scenarios where the dictionary is incomplete (i.e., entries for all word types are not present in the dictionary) and rare words

Method	Tagging accuracy (%)	Number of unique tag bigrams in final tagging output
EM	83.4	1195
IP	88.0	875
MIN-GREEDY	88.0	880

Figure 7: Results for unsupervised Italian POS tagging with a dictionary using a set of 90 tags.

take on all tag labels. In such cases, we can follow a similar approach as Ravi and Knight (2009) to assign tag possibilities to every unknown word using information from the known word/tag pairs present in the dictionary. Once the completed dictionary is available, we can use the procedure described in Section 5 to minimize the size of the grammar, followed by EM training.

6.2 Italian POS Tagging

We also compare the three approaches for Italian POS tagging and show results.

Data: We use the Italian CCG-TUT corpus (Bos et al., 2009), which contains 1837 sentences. It has three sections: newspaper texts, civil code texts and European law texts from the JRC-Acquis Multilingual Parallel Corpus. For our experiments, we use the POS-tagged data from the CCG-TUT corpus, which uses a set of 90 tags. We created a tag dictionary consisting of 8,733 word/tag pairs derived from the entire corpus (42,100 word tokens). We then created a test set consisting of 926 sentences (21,878 word tokens) from the original corpus. The per-token ambiguity for the test data is about 1.6 tags/token.

Results: Figure 7 shows the results on Italian POS tagging. We observe that MIN-GREEDY achieves significant improvements in tagging accuracy over the EM method and comparable to IP method. This also shows that the idea of model minimization is a general-purpose technique for such applications and provides good tagging accuracies on other languages as well.

7 Conclusion

We present a fast and efficient two-stage greedy minimization approach that can replace the integer programming step in (Ravi and Knight, 2009). The greedy approach finds close-to-optimal solutions for the minimization problem. Our algo-

rithm runs much faster and achieves accuracies close to state-of-the-art. We also evaluate our method on test sets of varying sizes and show that our approach outperforms standard EM by a significant margin. For future work, we would like to incorporate some linguistic constraints within the greedy method. For example, we can assign higher costs to unlikely tag combinations (such as “SYM SYM”, etc.).

Our greedy method can also be used for solving other unsupervised tasks where model minimization using integer programming has proven successful, such as word alignment (Bodrumlu et al., 2009).

Acknowledgments

The authors would like to thank Shang-Hua Teng and Anup Rao for their helpful comments and also the anonymous reviewers. This work was jointly supported by NSF grant IIS-0904684, DARPA contract HR0011-06-C-0022 under subcontract to BBN Technologies and DARPA contract HR0011-09-1-0028.

References

- Bodrumlu, T., K. Knight, and S. Ravi. 2009. A new objective function for word alignment. In *Proceedings of the NAACL/HLT Workshop on Integer Programming for Natural Language Processing*.
- Bos, J., C. Bosco, and A. Mazzei. 2009. Converting a dependency treebank to a categorial grammar treebank for Italian. In *Proceedings of the Eighth International Workshop on Treebanks and Linguistic Theories (TLT8)*.
- Clarke, J. and M. Lapata. 2008. Global inference for sentence compression: An integer linear programming approach. *Journal of Artificial Intelligence Research (JAIR)*, 31(4):399–429.
- Dempster, A.P., N.M. Laird and D.B. Rubin. 1977. Maximum likelihood from incomplete data via the

- EM algorithm. *Journal of the Royal Statistical Society*, 39(1):1–38.
- Garey, M. R. and D. S. Johnson. 1979. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. John Wiley & Sons.
- Goldberg, Y., M. Adler, and M. Elhadad. 2008. EM can find pretty good HMM POS-taggers (when given a good start). In *Proceedings of the 46th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies (ACL/HLT)*.
- Goldwater, Sharon and Thomas L. Griffiths. 2007. A fully Bayesian approach to unsupervised part-of-speech tagging. In *Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics (ACL)*.
- Martins, A., N. A. Smith, and E. P. Xing. 2009. Concise integer linear programming formulations for dependency parsing. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the Association for Computational Linguistics (ACL) and the 4th International Joint Conference on Natural Language Processing of the AFNLP*.
- Merialdo, B. 1994. Tagging English text with a probabilistic model. *Computational Linguistics*, 20(2):155–171.
- Punyakanok, V., D. Roth, W. Yih, and D. Zimak. 2004. Semantic role labeling via integer linear programming inference. In *Proceedings of the International Conference on Computational Linguistics (COLING)*.
- Ravi, S. and K. Knight. 2008. Attacking decipherment problems optimally with low-order n-gram models. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*.
- Ravi, S. and K. Knight. 2009. Minimized models for unsupervised part-of-speech tagging. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the Association for Computational Linguistics (ACL) and the 4th International Joint Conference on Natural Language Processing of the AFNLP*.
- Smith, N. and J. Eisner. 2005. Contrastive estimation: Training log-linear models on unlabeled data. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL)*.