

# A fast finite-state relaxation method for enforcing global constraints on sequence decoding

Roy W. Tromble and Jason Eisner

Department of Computer Science and Center for Language and Speech Processing

Johns Hopkins University

Baltimore, MD 21218

{royt, jason}@cs.jhu.edu

## Abstract

We describe finite-state constraint relaxation, a method for applying global constraints, expressed as automata, to sequence model decoding. We present algorithms for both hard constraints and binary soft constraints. On the CoNLL-2004 semantic role labeling task, we report a speedup of at least 16x over a previous method that used integer linear programming.

## 1 Introduction

Many tasks in natural language processing involve sequence labeling. If one models long-distance or global properties of labeled sequences, it can become intractable to find (“decode”) the best labeling of an unlabeled sequence.

Nonetheless, such global properties can improve the accuracy of a model, so recent NLP papers have considered practical techniques for decoding with them. Such techniques include Gibbs sampling (Finkel et al., 2005), a general-purpose Monte Carlo method, and integer linear programming (ILP), (Roth and Yih, 2005), a general-purpose exact framework for NP-complete problems.

Under generative models such as hidden Markov models, the probability of a labeled sequence depends only on its local properties. The situation improves with discriminatively trained models, such as conditional random fields (Lafferty et al., 2001), which do efficiently allow features that are functions of the entire *observation* sequence. However, these features can still only look locally at the *label* sequence. That is a significant shortcoming, because in many domains, hard or soft global constraints on the label sequence are motivated by common sense:

- For named entity recognition, a phrase that appears multiple times should tend to get the same label each time (Finkel et al., 2005).
- In bibliography entries (Peng and McCallum, 2004), a given field (author, title, etc.) should

be filled by at most one substring of the input, and there are strong preferences on the co-occurrence and order of certain fields.

- In seminar announcements, a given field (speaker, start time, etc.) should appear with at most one value in each announcement, although the field and value may be repeated (Finkel et al., 2005).
- For semantic role labeling, each argument should be instantiated only once for a given verb. There are several other constraints that we will describe later (Roth and Yih, 2005).

A popular approximate technique is to hypothesize a list of possible answers by decoding without any global constraints, and then *rerank* (or *prune*) this *n*-best list using the full model with all constraints. Reranking relies on the local model being “good enough” that the globally best answer appears in its *n*-best list. Otherwise, reranking can’t find it.

In this paper, we propose “constraint relaxation,” a simple exact alternative to reranking. As in reranking, we start with a weighted lattice of hypotheses proposed by the local model. But rather than restrict to the *n* best of these according to the local model, we aim to directly extract the *one* best according to the *global* model. As in reranking, we hope that the local constraints alone will work well, but if they do not, the penalty is not incorrect decoding, but longer runtime as we gradually fold the global constraints into the lattice. Constraint relaxation can be used whenever the global constraints can be expressed as regular languages over the label sequence.

In the worst case, our runtime may be exponential in the number of constraints, since we are considering an intractable class of problems. However, we show that in practice, the method is quite effective at rapid decoding under global hard constraints.

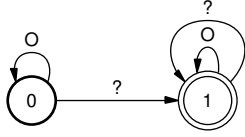


Figure 1: An automaton expressing the constraint that the label sequence cannot be  $0^*$ . Here  $?$  matches any symbol except 0.

The remainder of the paper is organized as follows: In §2 we describe how finite-state automata can be used to apply global constraints. We then give a brute-force decoding algorithm (§3). In §4, we present a more efficient algorithm for the case of hard constraints. We report results for the semantic role labeling task in §5. §6 treats soft constraints.

## 2 Finite-state constraints

Previous approaches to global sequence labeling—Gibbs sampling, ILP, and reranking—seem motivated by the idea that standard sequence methods are incapable of considering global constraints at all.

In fact, finite-state automata (FSAs) are powerful enough to express many long-distance constraints. Since all finite languages are regular, *any* constraint over label sequences of bounded length is finite-state. FSAs are more powerful than  $n$ -gram models. For example, the regular expression  $\Sigma^*X\Sigma^*Y\Sigma^*$  matches only sequences of labels that contain an  $X$  before a  $Y$ . Similarly, the regular expression  $\neg(0^*)$  requires at least one non-0 label; it compiles into the FSA of Figure 1.

Note that this FSA is in one or the other of its two states according to whether it has encountered a non-0 label yet. In general, the current state of an FSA records properties of the label sequence prefix read so far. The FSA needs enough states to keep track of whether the label sequence as a whole satisfies the global constraint in question.

FSAs are a flexible approach to constraints because they are closed under logical operations such as disjunction (union) and conjunction (intersection). They may be specified by regular expressions (Karttunen et al., 1996), in a logical language (Vaillette, 2004), or directly as FSAs. They may also be weighted to express soft constraints.

Formally, we pose the decoding problem in terms of an observation sequence  $\mathbf{x} \in \mathcal{X}^*$  and possible la-

bel sequences  $\mathbf{y} \in \mathcal{Y}^*$ . In many NLP tasks,  $\mathcal{X}$  is the set of words, and  $\mathcal{Y}$  the tags. A lattice  $L: \mathcal{Y}^* \mapsto \mathbb{R}$  maps label sequences to weights, and is encoded as a weighted FSA. Constraints are formally the same—any function  $C: \mathcal{Y}^* \mapsto \mathbb{R}$  is a constraint, including weighted features from a classifier or probabilistic model. In this paper we will consider only constraints that are weighted in particular ways.

Given a lattice  $L$  and constraints  $\mathcal{C}$ , we seek

$$\mathbf{y}^* \stackrel{\text{def}}{=} \operatorname{argmax}_{\mathbf{y}} \left( L(\mathbf{y}) + \sum_{C \in \mathcal{C}} C(\mathbf{y}) \right). \quad (1)$$

We assume the lattice  $L$  is generated by a model  $M: \mathcal{X}^* \mapsto (\mathcal{Y}^* \mapsto \mathbb{R})$ . For a given observation sequence  $\mathbf{x}$ , we put  $L = M(\mathbf{x})$ . One possible model is a finite-state transducer, where  $M(\mathbf{x})$  is an FSA found by composing the transducer with  $\mathbf{x}$ . Another is a CRF, where  $M(\mathbf{x})$  is a lattice with sums of log-potentials for arc weights.<sup>1</sup>

## 3 A brute-force finite-state decoder

To find the best constrained labeling in a lattice,  $\mathbf{y}^*$ , according to (1), we could simply intersect the lattice with all the constraints, then extract the best path.

Weighted FSA intersection is a generalization of ordinary unweighted FSA intersection (Mohri et al., 1996). It is customary in NLP to use the so-called tropical semiring, where weights are represented by their natural logarithms and summed rather than multiplied. Then the intersected automaton  $L \cap C$  computes

$$(L \cap C)(\mathbf{y}) \stackrel{\text{def}}{=} L(\mathbf{y}) + C(\mathbf{y}) \quad (2)$$

To find  $\mathbf{y}^*$ , one would extract the best path in  $L \cap C_1 \cap C_2 \cap \dots$  using the Viterbi algorithm, or Dijkstra’s algorithm if the lattice is cyclic. This step is fast if the intersected automaton is small.

The problem is that the multiple intersections in  $L \cap C_1 \cap C_2 \cap \dots$  can quickly lead to an FSA with an intractable number of states. The intersection of two finite-state automata produces an automaton

<sup>1</sup>For example, if  $M$  is a simple linear-chain CRF,  $L(\mathbf{y}) = \sum_{i=1}^n f(y_{i-1}, y_i) + g(x_i, y_i)$ . We build  $L = M(\mathbf{x})$  as an acyclic FSA whose state set is  $\mathcal{Y} \times \{1, 2, \dots, n\}$ , with transitions  $(y', i-1) \rightarrow (y, i)$  of weight  $f(y', y) + g(x_i, y)$ .

with the cross product state set. That is, if  $F$  has  $m$  states and  $G$  has  $n$  states, then  $F \cap G$  has up to  $mn$  states (fewer if some of the  $mn$  possible states do not lie on any accepting path).

Intersection of many such constraints, even if they have only a few states each, quickly leads to a combinatorial explosion. In the worst case, the size, in states, of the resulting lattice is exponential in the number of constraints. To deal with this, we present a constraint relaxation algorithm.

## 4 Hard constraints

The simplest kind of constraint is the hard constraint. Hard constraints are necessarily binary—either the labeling satisfies the constraint, or it violates it. Violation is fatal—the labeling produced by decoding must satisfy each hard constraint.

Formally, a hard constraint is a mapping  $C: \mathcal{Y}^* \mapsto \{0, -\infty\}$ , encoded as an unweighted FSA. If a string satisfies the constraint, recognition of the string will lead to an accepting state. If it violates the constraint, recognition will end in a non-accepting state.

Here we give an algorithm for decoding with a set of such constraints. Later (§6), we discuss the case of binary soft constraints. In what follows, we will assume that there is always at least one path in the lattice that satisfies all of the constraints.

### 4.1 Decoding by constraint relaxation

Our decoding algorithm first relaxes the global constraints and solves a simpler problem. In particular, we find the best labeling according to the model,

$$\mathbf{y}_0^* \stackrel{\text{def}}{=} \underset{\mathbf{y}}{\operatorname{argmax}} L(\mathbf{y}) \quad (3)$$

ignoring *all* the constraints in  $\mathcal{C}$ .

Next, we check whether  $\mathbf{y}_0^*$  satisfies the constraints. If so, then we are done— $\mathbf{y}_0^*$  is also  $\mathbf{y}^*$ . If not, then we reintroduce the constraints. However, rather than include all at once, we introduce them only as they are violated by successive solutions to the relaxed problems:  $\mathbf{y}_0^*$ ,  $\mathbf{y}_1^*$ , etc. We define

$$\mathbf{y}_1^* \stackrel{\text{def}}{=} \underset{\mathbf{y}}{\operatorname{argmax}} (L(\mathbf{y}) + C(\mathbf{y})) \quad (4)$$

for some constraint  $C$  that  $\mathbf{y}_0^*$  violates. Similarly,  $\mathbf{y}_2^*$  satisfies an additional constraint that  $\mathbf{y}_1^*$  violates,

```

HARD-CONSTRAIN-LATTICE( $L, \mathcal{C}$ ):
1.  $\mathbf{y} := \text{Best-Path}(L)$ 
2. while  $\exists C \in \mathcal{C}$  such that  $C(\mathbf{y}) = -\infty$ :
3.    $L := L \cap C$ 
4.    $\mathcal{C} := \mathcal{C} - \{C\}$ 
5.    $\mathbf{y} := \text{Best-Path}(L)$ 
6. return  $\mathbf{y}$ 

```

Figure 2: Hard constraints decoding algorithm.

and so on. Eventually, we find some  $k$  for which  $\mathbf{y}_k^*$  satisfies all constraints, and this path is returned.

To determine whether a labeling  $\mathbf{y}$  satisfies a constraint  $C$ , we represent  $\mathbf{y}$  as a straight-line automaton and intersect with  $C$ , checking the result for non-emptiness. This is equivalent to string recognition.

Our hope is that, although intractable in the worst case, the constraint relaxation algorithm will operate efficiently in practice. The success of traditional sequence models on NLP tasks suggests that, for natural language, much of the correct analysis can be recovered from local features and constraints alone. We suspect that, as a result, global constraints will often be easy to satisfy.

Pseudocode for the algorithm appears in Figure 2. Note that line 2 does *not* specify how to choose  $C$  from among multiple violated constraints. This is discussed in §7. Our algorithm resembles the method of Koskenniemi (1990) and later work. The difference is that there lattices are unweighted and may not contain a path that satisfies all constraints, so that the order of constraint intersection matters.

## 5 Semantic role labeling

The semantic role labeling task (Carreras and Màrques, 2004) involves choosing instantiations of verb arguments from a sentence for a given verb. The verb and its arguments form a *proposition*. We use data from the CoNLL-2004 shared task—the PropBank (Palmer et al., 2005) annotations of the Penn Treebank (Marcus et al., 1993), with sections 15–18 as the training set and section 20 as the development set. Unless otherwise specified, all measurements are made on the development set.

We follow Roth and Yih (2005) exactly, in order to compare system runtimes. They, in turn, follow Hacioglu et al. (2004) and others in labeling only the heads of syntactic chunks rather than all words. We label only the core arguments (A0–A5), treating

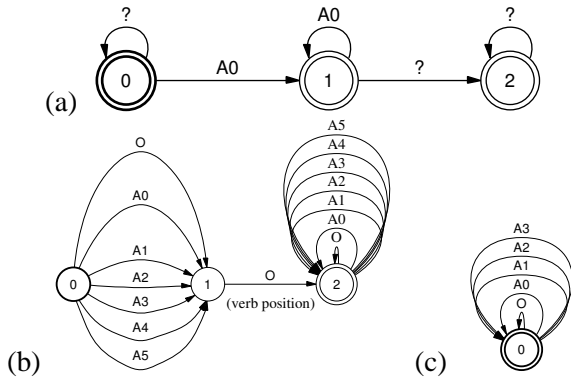


Figure 4: Automata expressing NO DUPLICATE A0 (? matches anything but A0), KNOWN VERB POSITION[2], and DISALLOW ARGUMENTS[A4,A5].

adjuncts and references as O.

Figure 3 shows an example sentence from the shared task. It is marked with an IOB phrase chunking, the heads of the phrases, and the correct semantic role labeling. Heads are taken to be the rightmost words of chunks. On average, there are 18.8 phrases per proposition, vs. 23.5 words per sentence. Sentences may contain multiple propositions. There are 4305 propositions in section 20.

## 5.1 Constraints

Roth and Yih use five global constraints on label sequences for the semantic role labeling task. We express these constraints as FSAs. The first two are general, and the seven automata encoding them can be constructed offline:

- NO DUPLICATE ARGUMENT LABELS (Fig. 4(a)) requires that each verb have at most one argument of each type in a given sentence. We separate this into six individual constraints, one for each core argument type. Thus, we have constraints called NO DUPLICATE A0, NO DUPLICATE A1, etc. Each of these is represented as a three-state FSA.
- AT LEAST ONE ARGUMENT (Fig. 1) simply requires that the label sequence is not  $0^*$ . This is a two-state automaton as described in §2.

The last three constraints require information about the example, and the automata must be constructed on a per-example basis:

- ARGUMENT CANDIDATES (Fig. 5) encodes a set of position spans each of which must receive only a single label type. These spans were proposed using a high-recall heuristic (Xue and Palmer, 2004).
- KNOWN VERB POSITION (Fig. 4(b)) simply encodes the position of the verb in question, which must be labeled O.
- DISALLOW ARGUMENTS (Fig. 4(c)) specifies argument types that are compatible with the verb in question, according to PropBank.

## 5.2 Experiments

We implemented our hard constraint relaxation algorithm, using the FSA toolkit (Kanthak and Ney, 2004) for finite-state operations. FSA is an open-source C++ library providing a useful set of algorithms on weighted finite-state acceptors and transducers. For each example we decoded, we chose a random order in which to apply the constraints.

Lattices are generated from what amounts to a unigram model—the voted perceptron classifier of Roth and Yih. The features used are a subset of those commonly applied to the task.

Our system produces output identical to that of Roth and Yih. Table 1 shows F-measure on the core arguments. Table 2 shows a runtime comparison. The ILP runtime was provided by the authors (personal communication). Because the systems were run under different conditions, the times are not directly comparable. However, constraint relaxation is more than sixteen times faster than ILP despite running on a slower platform.

### 5.2.1 Comparison to an ILP solver

Roth and Yih’s linear program has two kinds of numeric constraints. Some encode the shortest path problem structure; the others encode the global constraints of §5.1. The ILP solver works by relaxing to a (real-valued) linear program, which may obtain a fractional solution that represents a path mixture instead of a path. It then uses branch-and-bound to seek the optimal rounding of this fractional solution to an integer solution (Guéret et al., 2002) that represents a single path satisfying the global constraints.

Our method avoids fractional solutions: a relaxed solution is always a true single path, which either

Mr. Turner said the test will be shipped in 45 days to hospitals and clinical laboratories .  
 B-NP I-NP B-VP B-NP I-NP B-VP I-VP I-VP B-PP B-NP I-NP B-PP B-NP O B-NP I-NP O  
 Turner said test shipped in days to hospitals and laboratories .  
 A0 O A1 A1 A1 A1 A1 A1 A1 A1

Figure 3: Example sentence, with phrase tags and heads, and core argument labels. The A1 argument of “said” is a long clause.

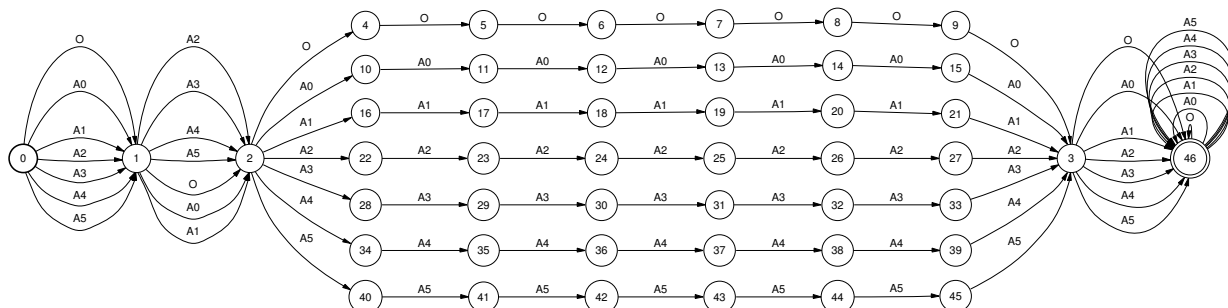


Figure 5: An automaton expressing ARGUMENT CANDIDATES.

Argument	Count	F-measure
A0	2849	79.27
A1	4029	75.59
A2	943	55.68
A3	149	46.41
A4	147	81.82
A5	4	25.00
All	8121	74.51

Table 1: F-measure on core arguments.

Constraint	Violations	Fraction
ARGUMENT CANDIDATES	1619	0.376
NO DUPLICATE A1	899	0.209
NO DUPLICATE A0	348	0.081
NO DUPLICATE A2	151	0.035
AT LEAST ONE ARGUMENT	108	0.025
DISALLOW ARGUMENTS	48	0.011
NO DUPLICATE A3	13	0.003
NO DUPLICATE A4	3	0.001
NO DUPLICATE A5	1	0.000
KNOWN VERB POSITION	0	0.000

Table 3: Violations of constraints by  $y_0^*$ .

satisfies or violates each global constraint. In effect, we are using two kinds of domain knowledge. First, we recognize that this is a graph problem, and insist on true paths so we can use Viterbi decoding. Second, we choose to relax only domain-specific constraints that are likely to be satisfied anyway (in our domain), in contrast to the meta-constraint of integrality relaxed by ILP. Thus it is cheaper on average for us to repair a relaxed solution. (Our repair strategy—finite-state intersection in place of branch-and-bound search—remains expensive in the worst case, as the problem is NP-hard.)

### 5.2.2 Constraint violations

The  $y_0^*$ s, generated with only *local* information, satisfy most of the global constraints most of the time. Table 3 shows the violations by type.

The *majority* of best labelings according to the local model don’t violate *any* global constraints—a fact especially remarkable because there are *no* label sequence features in Roth and Yih’s unigram

model. This confirms our intuition that natural language structure is largely apparent locally. Table 4 shows the breakdown. The majority of examples are very efficient to decode, because they don’t require intersection of the lattice with any constraints— $y_0^*$  is extracted and is good enough. Those examples where constraints *are* violated are still relatively efficient because they only require a small number of intersections. In total, the average number of intersections needed, even with the naive randomized constraint ordering, was only 0.65. The order doesn’t matter very much, since 75% of examples have one violation or fewer.

### 5.2.3 Effects on lattice size

Figure 6 shows the effect of intersection with violated constraints on the average size of lattices, measured in arcs. The vertical bars at  $k = 0, k = 1, \dots$  show the number of examples where con-

Method	Total Time	Per Example	Platform
Brute Force Finite-State	37m25.290s	0.522s	Pentium III, 1.0 GHz
ILP	11m39.220s	0.162s	Xeon, 3.x GHz
Constraint Relaxation	39.700s	0.009s	Pentium III, 1.0 GHz

Table 2: A comparison of runtimes for constrained decoding with ILP and FSA.

Violations	Labelings	Fraction	Cumulative
0	2368	0.550	0.550
1	863	0.200	0.750
2	907	0.211	0.961
3	156	0.036	0.997
4	10	0.002	0.999
5	1	0.000	1.000
6–10	0	0.000	1.000

Table 4: Number of  $\mathbf{y}_0^*$  with each violation count.

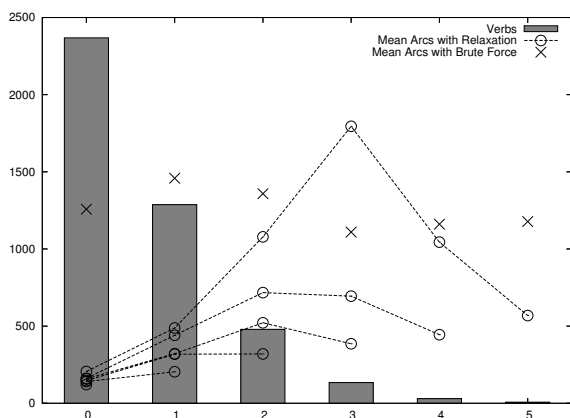


Figure 6: Mean lattice size (measured in arcs) throughout decoding. Vertical bars show the number of examples over which each mean is computed.

straint relaxation had to intersect  $k$  constraints (i.e.,  $\mathbf{y}^* \equiv \mathbf{y}_k^*$ ). The trajectory ending at (for example)  $k = 3$  shows how the average lattice size for that subset of examples evolved over the 3 intersections. The X at  $k = 3$  shows the final size of the brute-force lattice on the *same* subset of examples.

For the most part, our lattices do stay much smaller than those produced by the brute-force algorithm. (The uppermost curve,  $k = 5$ , is an obvious exception; however, that curve describes only the seven hardest examples.) Note that plotting only the final size of the brute-force lattice obscures the long trajectory of its construction, which involves 10 intersections and, like the trajectories shown, includes larger intermediate automata.<sup>2</sup> This explains the far

<sup>2</sup>The final brute-force lattice is especially shrunk by its in-

Constraint	Violations	Fraction
ARGUMENT CANDIDATES	90	0.0209
AT LEAST ONE ARGUMENT	27	0.0063
NO DUPLICATE A2	3	0.0007
NO DUPLICATE A0	2	0.0005
NO DUPLICATE A1	2	0.0005
NO DUPLICATE A3	1	0.0002
NO DUPLICATE A4	1	0.0002

Table 5: Violations of constraints by  $\hat{\mathbf{y}}$ , measured over the development set.

longer runtime of the brute-force method (Table 2).

Harder examples (corresponding to longer trajectories) have larger lattices, on average. This is partly just because it is disproportionately the longer sentences that are hard: they have more opportunities for a relaxed decoding to violate global constraints.

Hard examples are rare. The left three columns, requiring only 0–2 intersections, constitute 96% of examples. The vast majority can be decoded without much more than doubling the local-lattice size.

## 6 Soft constraints

The gold standard labels  $\hat{\mathbf{y}}$  occasionally violate the hard global constraints that we are using. Counts for the development set appear in Table 5. Counts for violations of NO DUPLICATE A· do *not* include discontinuous arguments, of which there are 104 instances, since we ignore them.

Because of the infrequency, the hard constraints still help most of the time. However, on a small subset of the examples, they preclude us from inferring the correct labeling.

We can apply these constraints with weights, rather than making them inviolable. This constitutes a transition from hard to soft constraints. Formally, a soft constraint  $C: \mathcal{Y}^* \mapsto \mathbb{R}^-$  is a mapping from a label sequence to a non-positive penalty.

Soft constraints present new difficulty for decoding of, for example, DISALLOW ARGUMENTS, which can only remove arcs. That constraint is rarely included in the relaxation lattices because it is rarely violated (see Table 3).

```

SOFT-CONSTRAIN-LATTICE( $L, C$ ):
1.  $(\mathbf{y}^*, \text{Score}(\mathbf{y}^*)) := (\text{empty}, -\infty)$ 
2.  $\text{branches} := [(L, C, 0)]$ 
3. while  $(L, C, \text{penalty}) := \text{Dequeue}(\text{branches})$ :
4.    $L := \text{Prune}(L, \text{Score}(\mathbf{y}^*) - \text{penalty})$ 
5.   unless  $\text{Empty}(L)$ :
6.      $\mathbf{y} := \text{Best-Path}(L)$ 
7.     for  $C \in \mathcal{C}$ :
8.       if  $C(\mathbf{y}) < 0$ : (* so  $C(\mathbf{y}) = w_C$  *)
9.          $C := C - \{C\}$ 
10.         $\text{Enqueue}(\text{branches}, (L \cap C, C, \text{penalty}))$ 
11.         $\text{penalty} := \text{penalty} + C(\mathbf{y})$ 
12.       if  $\text{Score}(\mathbf{y}^*) < L(\mathbf{y}) + \text{penalty}$ :
13.          $(\mathbf{y}^*, \text{Score}(\mathbf{y}^*)) := (\mathbf{y}, L(\mathbf{y}) + \text{penalty})$ 
14. return  $\mathbf{y}^*$ 

```

Figure 7: Soft constraints decoding algorithm

ing, because instead of eliminating paths of  $L$  from contention, they just reweight them.

In what follows, we consider only binary soft constraints—they are either satisfied or violated, and the same penalty is assessed whenever a violation occurs. That is,  $\forall C \in \mathcal{C}, \exists w_C < 0$  such that  $\forall \mathbf{y}, C(\mathbf{y}) \in \{0, w_C\}$ .

### 6.1 Soft constraint relaxation

The decoding algorithm for soft constraints is a generalization of that for hard constraints. The difference is that, whereas with hard constraints a violation meant disqualification, here violation simply means a penalty. We therefore must find and compare two labelings: the best that satisfies the constraint, and the best that violates it.

We present a branch-and-bound algorithm (Lawler and Wood, 1966), with pseudocode in Figure 7. At line 9, we process and eliminate a currently violated constraint  $C \in \mathcal{C}$  by considering two cases. On the first branch, we insist that  $C$  be satisfied, enqueueing  $L \cap C$  for later exploration. On the second branch, we assume  $C$  is violated by all paths, and so continue considering  $L$  unmodified, but accept a penalty for doing so; we immediately explore the second branch by returning to the start of the **for** loop.<sup>3</sup>

Not every branch needs to be completely explored. Bounding is handled by the PRUNE function at line 4, which shrinks  $L$  by removing some

<sup>3</sup>It is possible that a future best path on the second branch will *not* actually violate  $C$ , in which case we have overpenalized it, but in that case we will also find it with correct penalty on the first branch.

or all paths that cannot score better than  $\text{Score}(\mathbf{y}^*)$ , the score of the best path found on any branch so far. Our experiments used almost the simplest possible PRUNE: replace  $L$  by the empty lattice if the best path falls below the bound, else leave  $L$  unchanged.<sup>4</sup>

A similar bounding would be possible in the implicit branches. If, during the **for** loop, we find that the test at line 12 would fail, we can quit the **for** loop and immediately move to the next branch in the queue at line 3.

There are two factors in this algorithm that contribute to avoiding consideration of *all* of the exponential number of leaves corresponding to the power set of constraints. First, bounding stops evaluation of subtrees. Second, only *violated* constraints require branching. If a lattice’s best path satisfies a constraint, then the best path that violates it can be no better since, by assumption,  $\forall \mathbf{y}, C(\mathbf{y}) \leq 0$ .

### 6.2 Runtime experiments

Using the ten constraints from §5.1, weighted naively by their log odds of violation, the soft constraint relaxation algorithm runs in a time of 58.40 seconds. It is, as expected, slower than hard constraint relaxation, but only by a factor of about two.

As a side note, softening these particular constraints in this particular way did not improve decoding quality in this case. It might help to jointly train the relative weights of these constraints and the local model—e.g., using a perceptron algorithm (Freund and Schapire, 1998), which repeatedly extracts the best global path (using our algorithm), compares it to the gold standard, and adjusts the constraint weights. An obvious alternative is maximum-entropy training, but the partition function would have to be computed using the large brute-force lattices, or else approximated by a sampling method.

## 7 Future work

For a given task, we may be able to obtain further speedups by carefully choosing the order in which to test and apply the constraints. We might treat this as a reinforcement learning problem (Sutton, 1988),

<sup>4</sup>Partial pruning is also possible: by running the Viterbi version of the forward-backward algorithm, one can discover for each edge the weight of the best path on which it appears. One can then remove all edges that do not appear on any sufficiently good path.

where an agent will obtain rewards by finding  $y^*$  quickly. In the hard-constraint algorithm, for example, the agent's possible moves are to test some constraint for violation by the current best path, or to intersect some constraint with the current lattice. Several features can help the agent choose the next move. How large is the current lattice, which constraints does it already incorporate, and which remaining constraints are already known to be satisfied or violated by its best path? And what were the answers to those questions at previous stages?

Our constraint relaxation method should be tested on problems other than semantic role labeling. For example, information extraction from bibliography entries, as discussed in §1, has about 13 fields to extract, and interesting hard and soft global constraints on co-occurrence, order, and adjacency. The method should also be evaluated on a task with longer sequences: though the finite-state operations we use do scale up linearly with the sequence length, longer sequences have more chance of violating a global constraint somewhere in the sequence, requiring us to apply that constraint explicitly.

## 8 Conclusion

Roth and Yih (2005) showed that global constraints can improve the output of sequence labeling models for semantic role labeling. In general, decoding under such constraints is NP-complete. We exhibited a practical approach, finite-state constraint relaxation, that greatly sped up decoding on this NLP task by using familiar finite-state operations—weighted FSA intersection and best-path extraction—rather than integer linear programming.

We have also given a constraint relaxation algorithm for binary soft constraints. This allows incorporation of constraints akin to reranking features, in addition to inviolable constraints.

## Acknowledgments

This material is based upon work supported by the National Science Foundation under Grant No. 0347822. We thank Scott Yih for kindly providing both the voted-perceptron classifier and runtime results for decoding with ILP, and the reviewers for helpful comments.

## References

- Xavier Carreras and Lluís Màrques. 2004. Introduction to the CoNLL-2004 shared task: Semantic role labeling. In *Proc. of CoNLL*, pp. 89–97.
- Jenny Rose Finkel, Trond Grenager, and Christopher Manning. 2005. Incorporating non-local information into information extraction systems by Gibbs sampling. In *Proc. of ACL*, pp. 363–370.
- Yoav Freund and Robert E. Schapire. 1998. Large margin classification using the perceptron algorithm. In *Proc. of COLT*, pp. 209–217, New York. ACM Press.
- Christelle Guéret, Christian Prins, and Marc Sevaux. 2002. *Applications of optimization with Xpress-MP*. Dash Optimization. Translated and revised by Susanne Heipcke.
- Kadri Hacioglu, Sameer Pradhan, Wayne Ward, James H. Martin, and Daniel Jurafsky. 2004. Semantic role labeling by tagging syntactic chunks. In *Proc. of CoNLL*, pp. 110–113.
- Stephan Kanthak and Hermann Ney. 2004. FSA: An efficient and flexible C++ toolkit for finite state automata using on-demand computation. In *Proc. of ACL*, pp. 510–517.
- Lauri Karttunen, Jean-Pierre Chanod, Gregory Grefenstette, and Anne Schiller. 1996. Regular expressions for language engineering. *Journal of Natural Language Engineering*, 2(4):305–328.
- Kimmo Koskenniemi. 1990. Finite-state parsing and disambiguation. In *Proc. of COLING*, pp. 229–232.
- John Lafferty, Andrew McCallum, and Fernando Pereira. 2001. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proc. of ICML*, pp. 282–289.
- Eugene L. Lawler and David E. Wood. 1966. Branch-and-bound methods: A survey. *Operations Research*, 14(4):699–719.
- Mitchell P. Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. 1993. Building a large annotated corpus of English: the Penn Treebank. *Computational Linguistics*, 19:313–330.
- Mehryar Mohri, Fernando Pereira, and Michael Riley. 1996. Weighted automata in text and speech processing. In A. Kornai, editor, *Proc. of the ECAI 96 Workshop*, pp. 46–50.
- Martha Palmer, Daniel Gildea, and Paul Kingsbury. 2005. The Proposition Bank: An annotated corpus of semantic roles. *Computational Linguistics*, 31(1):71–106.
- Fuchun Peng and Andrew McCallum. 2004. Accurate information extraction from research papers using conditional random fields. In *Proc. of HLT-NAACL*, pp. 329–336.
- Dan Roth and Wen-tau Yih. 2005. Integer linear programming inference for conditional random fields. In *Proc. of ICML*, pp. 737–744.
- Richard S. Sutton. 1988. Learning to predict by the methods of temporal differences. *Machine Learning*, 3(1):9–44.
- Nathan Vaillette. 2004. *Logical Specification of Finite-State Transductions for Natural Language Processing*. Ph.D. thesis, Ohio State University.
- Nianwen Xue and Martha Palmer. 2004. Calibrating features for semantic role labeling. In *Proc. of EMNLP*, pp. 88–94.