Algorithms for Minimum Risk Chunking

Martin Jansche

Center for Computational Learning Systems
Columbia University, New York

Abstract. Stochastic finite automata are useful for identifying substrings (chunks) within larger units of text. Relevant applications include tokenization, base-NP chunking, named entity recognition, and other information extraction tasks. For a given input string, a stochastic automaton represents a probability distribution over strings of labels encoding the location of chunks. For chunking and extraction tasks, the quality of predictions is evaluated in terms of precision and recall of the chunked/extracted phrases when compared against some gold standard. However, traditional methods for estimating the parameters of a stochastic finite automaton and for decoding the best hypothesis do not pay attention to the evaluation criterion, which we take to be the well-known F-measure. We are interested in methods that remedy this situation, both in training and decoding. Our main result is a novel algorithm for efficiently evaluating expected F-measure. We present the algorithm and discuss its applications for utility/risk-based parameter estimation and decoding.

1 Introduction

Finding regions of interest in texts is a fundamental task in Natural Language Processing. Typical regions of interest include noun phrases [1–4], subject-verb phrases [5], named entities [6, 7], and word tokens [8], among others. We consider this task abstractly and speak of *chunks* or *phrases* to be located inside larger strings. Phrase chunking – the process of finding chunks/phrases – is evaluated like an information retrieval task, in terms of precision and recall: we compare the set of chunks found by a system against a given "gold standard" dataset annotated with chunk information. Precision refers to the number of true positive chunks divided by the number of hypothesized chunks (fraction correct). Recall refers to the number of true positive chunks divided by the number of true chunks according to the gold standard (fraction found). Precision and recall values are combined into a single quantity, either the risk-like E-measure [9], or the utility-like F-measure.

Our larger goal is to formulate a stochastic approach to phrase chunking that is informed by these evaluation criteria: we want to minimize E-measure (risk) or maximize F-measure (utility) during training and decoding. In this paper we focus on the foundational aspects of this approach and on algorithmic issues surrounding minimum-risk/maximum-utility estimation in particular. The main result is a novel algorithm for evaluating the expected utility of a hypothesis. The

key insight is that the number of true positives (matched chunks) of a hypothesis string compared to a gold standard label string can be computed by a weighted infinite transducer. This infinity poses no problems, since transducers can be implemented in a lazy fashion [10] and only finite prefixes have to be considered. The algorithm also makes use of weighted transducer composition and algebraic path computations (not discussed here).

Building automatic chunkers is simplified by the use of supervised machine learning. In this scenario, a learner is presented with examples of strings together with a set of chunks[1] occurring in those strings and asked to infer regularities that will allow similar chunks to be found inside previously unseen strings. Since this is not a standard learning task, it is transformed into a more conventional sequence learning task. Learning with sequential data is ubiquitous in Natural Language Processing and well understood [11–13].

Several reductions from the real learning task to sequence learning tasks are possible. The most common schemes annotate each string of words with an equally long string of labels, which indicate, directly or indirectly, whether a symbol is part of a chunk, and whether it occurs at the start, in the middle, or at the end of a chunk. Tjong Kim Sang et al. [4] compare several labeling schemes. The examples and techniques in this paper are based on what they refer to as the IOB2 scheme. This choice is convenient, but not essential; our techniques could be adapted to work with other schemes as well. The IOB2 scheme goes back to Ratnaparkhi ([14], pp. 57ff.), who used the labels 'Start', 'Join', and 'Other', which are known here as as B, I, and O, respectively. Their function is perhaps best illustrated by an example.

Example 1. The following sentence appears in the Dutch language training data provided for the Shared Task of the 2002 Workshop on Computational Natural Language Learning [15], in which named entity chunks are indicated by square brackets (we do not care about entity sorts here):

> De liberale minister van [Justitie] [Marc Verwilghen] is geen kandidaat op de lokale [VLD-lijst] bij de komende gemeenteraadsverkiezingen in [Dendermonde].[2]

The same sentence is represented in the IOB2 scheme as follows:

> ⟨De, O⟩ ⟨liberale, O⟩ ⟨minister, O⟩ ⟨van, O⟩ ⟨Justitie, B⟩ ⟨Marc, B⟩
> ⟨Verwilghen, I⟩ ⟨is, O⟩ ⟨geen, O⟩ ⟨kandidaat, O⟩ ⟨op, O⟩ ⟨de, O⟩ ⟨lokale, O⟩
> ⟨VLD-lijst, B⟩ ⟨bij, O⟩ ⟨de, O⟩ ⟨komende, O⟩
> ⟨gemeenteraadsverkiezingen, O⟩ ⟨in, O⟩ ⟨Dendermonde, B⟩ ⟨., O⟩

[1] In the simplest case, a chunk is a substring of the string of words. This is the definition we will assume throughout this paper. More complex scenarios where chunks come in different varieties are easily accommodated.

[2] [Marc Verwilghen], the liberal minister of [Justice], is not on the local [VLD (the Flemish liberal democrats) list] as a candidate in the upcoming city council elections in [Dendermonde].

In the IOB2 scheme, the label B signals the **b**eginning of a chunk, I marks the **i**nside (continuation) of a chunk, and O denotes that a word is **o**utside of any chunk. Note that an I label cannot occur immediately after an O label. A total of three types of labels is needed in order to encode adjacent chunks, as in 'minister van [Justitie] [Marc Verwilghen]'.

Formally, an unsupervised instance is a nonempty string $w \in \Sigma^+$ over some finite alphabet Σ (Dutch words, in the above example). Let $\Gamma = \{\text{I}, \text{O}, \text{B}\}$ be the set of IOB2 labels. A supervised instance is then a pair $\langle w, x \rangle$ consisting of a word string $w \in \Sigma^+$ of length $|w| = \ell > 0$ together with a label sequence x of the same length $|x| = \ell$. The language of valid label sequences, of which x is a member, is the local language $L_{\text{lbl}} = \{\text{O}, \text{B}\}\Gamma^* - \Gamma^*\{\text{OI}\}\Gamma^*$. Excluded from L_{lbl} are label strings that either start with the label I or contain OI as a substring. A pair $\langle w, x \rangle$ of same-length strings is isomorphic to a string of pairs, which was the representation used in the example.

2 Two Related Processing Tasks

2.1 Minimum Risk Decoding

Thanks to the IOB2 encoding of chunks, we are now dealing with a familiar sequence labeling problem: instead of finding chunks in an instance w, we have to find a label string x corresponding to w in the transformed problem. However, we also need to recover a solution to the original information extraction problem from a solution to the sequence labeling problem. This is known as *decoding*.

Say a sequence labeling module is presented with a word string $w \in \Sigma^n$ and produces a probability distribution over label string hypotheses $y \in \Gamma^n \cap L_{\text{lbl}}$. A naive decoding approach might consider only the most likely label string and read chunks off that string. This does not use information from runners-up, which might contradict the most likely string and collectively outweigh it.

The Bayes Decision Rule [16] tells us that the best hypothesis $\hat{x}_w$ is one with minimum average cost under the distribution of label strings (this is also known as *minimum risk decoding*):

$$\hat{x}(w) = \operatorname*{argmin}_x \mathcal{R}(x \mid w) = \operatorname*{argmin}_x \sum_y \lambda(x \mid y) \Pr(y \mid w), \tag{1}$$

where $x, y \in \Gamma^{|w|} \cap L_{\text{lbl}}$ range over valid label strings such that $|w| = |x| = |y|$. The *loss function* λ is a task-dependent function into the nonnegative rational numbers; $\lambda(x \mid y)$ is the loss incurred for choosing hypothesis x when the true state of affairs is y. Finally, $\mathcal{R}(x \mid w)$ is the *conditional risk*, or expected loss, of hypothesis x under a probability distribution conditional on w.

Instead of minimizing the expectation of a loss function, we can also maximize the expectation of the negative loss function, which we call the expected *utility*.[3] In general, the choice of loss or utility function depends on the application. In Natural Language Processing, various loss/utility functions have been

[3] Maximizing expected utility and minimizing expected loss amount to the same if optimization is exact. In the rest of this paper we will treat them as equivalent.

proposed for decoding and evaluation of chunking [5, 17–19] and other tasks (for example [20, 21] among many others). For the phrase chunking applications we are concerned with, the evaluation criteria are based on the concepts of precision and recall from Information Retrieval. Both of these criteria compare a hypothesis h against a gold standard g. Precision is defined as the number of correctly identified chunks (true positives, or $tp(g,h)$) divided by the number of hypothesized chunks (positive margin, or $m(h)$). We write this as

$$P(h \mid g) = \begin{cases} \dfrac{tp(g,h)}{m(h)} & \text{if } m(h) > 0 \\ 1 & \text{if } m(h) = 0 \end{cases} \tag{2}$$

Recall is conversely defined as the number of correctly identified chunks $tp(g,h)$ divided by the true number of chunks $m(g)$ (true margin): $R(h \mid g) = P(g \mid h)$. Note that a special case arises when the denominator is zero, in which case the numerator $tp(g,h)$ is also necessarily zero.

Precision and recall are combined into a single loss function, namely van Rijsbergen's ([9], p. 372) effectiveness measure E with parameter $\alpha \in (0;1)$:

$$E_\alpha(h \mid g) = 1 - \left[\alpha \frac{1}{P(h \mid g)} + (1-\alpha) \frac{1}{R(h \mid g)}\right]^{-1}$$

An analogous, and much more familiar, utility function can be defined in terms of $1 - E_\alpha(h \mid g)$. This is the α-weighted harmonic mean of precision and recall, also known as the F-measure and often mistakenly attributed to [22]. Letting $\alpha = 1/(\beta+1)$ with $\beta > 0$, the F_β-measure is defined as follows:

$$F_\beta(h \mid g) = 1 - E_\alpha(h \mid g) = \frac{(\beta+1)\, P(h \mid g)\, R(h \mid g)}{\beta\, P(h \mid g) + R(h \mid g)}$$

It is more convenient to express the F_β-measure in terms of the number of matched chunks $tp(g,h)$, hypothesized chunks $m(h)$, and actual chunks $m(g)$:

$$F_\beta(h \mid g) = \begin{cases} \dfrac{(\beta+1)\, tp(g,h)}{m(h) + \beta\, m(g)} & \text{if } m(h) + m(g) > 0 \\ 1 & \text{otherwise} \end{cases} \tag{3}$$

This equation gives us the parametric family of utility functions F_β that will be used throughout the rest of this paper for the phrase chunking task. The optimization tasks underlying decoding and parameter estimation involve maximizing the expectation of the utility $F_\beta(h \mid g)$, or, equivalently, minimizing the expectation of the loss $1 - F_\beta(h \mid g)$. Observe the following symmetry. Because $\alpha = 1/(\beta+1)$, therefore $1 - \alpha = \beta/(\beta+1) = 1/(1+1/\beta)$ and so

$$F_\beta(h \mid g) = F_{1/\beta}(g \mid h). \tag{4}$$

With these definitions in place, we return to the specifics of the decoding problem for the phrase chunking task. The hypothesis with minimum expected loss

(minimum risk) or maximum expected utility (MEU) is now

$$\hat{x}_{\text{MEU}}(w) = \operatorname*{argmax}_x \sum_y F_\beta(x \mid y)\ \Pr(y \mid w) = \operatorname*{argmax}_x \mathcal{U}_\beta(x \mid w). \tag{5}$$

The conditional expectation of F_β will also be written as $\mathcal{U}_\beta(x \mid w)$. In order to carry out the discrete optimization of the decoding task (5), we need an efficient algorithm for evaluating the expected utility $\mathcal{U}$. We will show that $\mathcal{U}$ can be represented as a weighted transducer whenever the probability model is provided by a stochastic finite automaton.

2.2 Parameter Estimation by Empirical Risk Minimization

A second task in which the expected utility $\mathcal{U}$ plays a role is the estimation of parameters of the underlying probability model of a chunker given a sequence of supervised instances $\langle w_1, x_1\rangle, \ldots, \langle w_n, x_n\rangle$. We assume that the probability model is parameterized by a vector θ. In Empirical Risk Minimization, our estimate of θ is one which minimizes the average risk on the training data. We reformulate this again as maximizing the average utility (using F_β as the utility function) instead of minimizing expected loss:

$$\hat{\theta} = \operatorname*{argmax}_\theta \frac{1}{n} \sum_{i=1}^{n} F_\beta(\hat{x}(w_i; \theta) \mid x_i).$$

For simplicity, we use as the decoded hypothesis $\hat{x}$ the maximum a posteriori (MAP) hypothesis (6) instead of the MEU hypothesis (5).

$$\hat{x}(w; \theta) = \operatorname*{argmax}_x \Pr(x \mid w; \theta). \tag{6}$$

The factor $1/n$ does not depend on θ and can be ignored in the maximization. The parameter estimation task is then the following optimization problem:

$$\hat{\theta} = \operatorname*{argmax}_\theta \sum_{i=1}^{n} F_\beta(\operatorname*{argmax}_z \Pr(z \mid w_i; \theta) \mid x_i). \tag{7}$$

Because of the nested discrete maximization step involving z, the outer maximization problem involving θ is not well-behaved: the outer optimization objective is a piecewise constant function of θ whose gradient is zero almost everywhere. We reformulate this problem and approximate the inner maximization in a way that will regularize the outer optimization problem. Let δ be the Kronecker delta, whose value is one if its two arguments are equal, and zero otherwise. Then:

$$\begin{aligned}
\hat{\theta} &= \operatorname*{argmax}_\theta \sum_{i=1}^{n} \sum_y F_\beta(y \mid x_i)\ \delta(y, \operatorname*{argmax}_z \Pr(z \mid w_i; \theta)) \\
&\approx \operatorname*{argmax}_\theta \sum_{i=1}^{n} \sum_y F_\beta(y \mid x_i)\ \frac{\Pr(y \mid w_i; \theta)}{\max_z \Pr(z \mid w_i; \theta)} \\
&= \operatorname*{argmax}_\theta \sum_{i=1}^{n} \frac{1}{\max_z \Pr(z \mid w_i; \theta)}\ \mathcal{U}_{1/\beta}(x_i \mid w_i; \theta)
\end{aligned}$$

The approximate equality between the first and second line holds because

$$\delta(y, \operatorname*{argmax}_z \Pr(z)) = \lim_{\gamma \to \infty} \left(\frac{\Pr(y)}{\max_z \Pr(z)} \right)^\gamma$$

We chose a fixed sharpening parameter $\gamma = 1$ to approximate this limit in the above derivation; larger values of γ can be used with minor changes. We again encounter the expected utility $\mathcal{U}$, whose parameter $1/\beta$ is due to the symmetry (4) observed earlier. The net result is that the outer optimization objective depends continuously on θ so that an iterative numerical optimization can be carried out[4] provided that $\mathcal{U}$ can be evaluated efficiently.

2.3 A Common Subexpression: Expected Utility

The conditional expected utility $\mathcal{U}$ occurs both in the Maximum Expected Utility (or Minimum Risk) decoding task and in the Empirical Risk Minimization parameter estimation task. It can be expressed as follows:

$$\mathcal{U}_\beta(x \mid w; \theta) = \begin{cases} (\beta + 1) \sum_y \dfrac{tp(x, y)}{m(x) + \beta\, m(y)} \Pr(y \mid w; \theta) & \text{if } m(x) > 0 \\ \Pr(x \mid w; \theta) & \text{if } m(x) = 0 \end{cases} \tag{8}$$

The special case is due to the fact that $m(x) + m(y) = 0$ iff both x and y are comprised exclusively of O labels, in which case $x = y$.

Expected utility cannot be evaluated efficiently by direct summation, since there are exponentially many label strings y one has to sum over. length ℓ (recall that $\ell = |w| = |x| = |y|$) be known as $N(\ell) = |\Gamma^\ell \cap L_{\text{lbl}}|$. It is easy to show that the asymptotic growth of N is exponential in ℓ: observe that $\{\text{O}, \text{B}\}^\ell \subsetneq (\Gamma^\ell \cap L_{\text{lbl}}) \subsetneq \Gamma^\ell$ for $\ell \geq 2$, and therefore $N(\ell) \in \omega(2^\ell)$ and $N(\ell) \in o(3^\ell)$. The tight bound is $\Theta((1 + \phi)^\ell)$ where $\phi = (1 + \sqrt{5})/2$; hence $1 + \phi \approx 2.618$. The hidden constant of proportionality in the tight bound is $1/2 + \sqrt{1/20}$, and so for a moderately long sentence with 21 words (including punctuation) like in Example 1, one would have to sum over 433 494 437 distinct label strings. The longest "sentence" in the dataset which Example 1 was taken from [15] is 859 words in length – it is a linearized table – and corresponds to about 79 centumoctodecillion (79×10^{357}) potential label strings.

3 Algorithms

3.1 Computing Expected Precision

Consider the problem of evaluating the expectation of precision (2) for fixed w, x and θ:

$$\sum_y P(x \mid y)\ \Pr(y \mid w; \theta) = \begin{cases} \dfrac{1}{m(x)} \sum_y tp(x, y)\ \Pr(y \mid w; \theta) & \text{if } m(x) > 0 \\ 1 & \text{if } m(x) = 0 \end{cases}$$

[4] This involves holding the factor $\max_z \Pr(z \mid w_i; \theta)$ constant in each iteration.

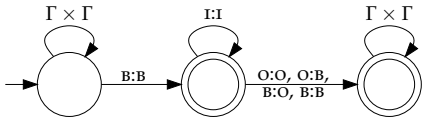

Fig. 1. Nondeterministic finite transducer that counts true positive chunks.

Computing expected precision boils down to evaluating the expected number of true positives,

$$\sum_y tp(x, y)\ \Pr(y \mid w; \theta). \tag{9}$$

The significance of (9) is that it also occurs in the derivation of the expected utility (8). The technique developed in this section for evaluating expected true positives will be generalized in Sec. 3.2 to apply to expected utility.

In general, sums of products of the form $\sum_y f(x, y)\, g(y, z)$ can be calculated efficiently for certain forms of f and g even when naive summation would be inefficient. This holds in particular when f and g can be computed by finite state transducers, in which case the summation corresponds to weighted transducer composition [23].

The expected number of true positives as expressed in (9) is of the requisite form. In order for transducer composition to be applicable, we need to show that we can compute tp by a finite state transducer.[5] Since we require the evaluation algorithm to be efficient, we also need to demonstrate that the size of all finite state machines involved in the computation is small enough to enable the evaluation to be carried out in polynomial time. We begin by formulating a finite state machine for computing tp.

A transducer that computes tp is a two-tape automaton that maps a pair of strings $x, y \in \Gamma^n$ to a count of the number of chunks that x and y agree on (true positives). An individual true positive chunk is described by the regular expression B:B (I:I)* ($:$ | O:O | O:B | B:O | B:B). This expression describes all pairs of string that both start with B followed by an equal amount of I's, and then both signal the end of the chunk. A chunk ends on either tape if the end of the string is reached ($), or if I is followed by a label other than I.

From this regular expression one can then construct a nondeterministic transducer that counts the number of occurrences of true positive chunks, using the generalized counting technique of Allauzen et al. [24]. The resulting weighted transducer over the real semiring, call it T_{nd}, is shown in Fig. 1 (all edge weights and final weights are one). The crucial observation is that $tp(x, y) = t$ if and only if there are precisely t paths through T_{nd} labeled with $\langle x, y \rangle$. For further background on weighted transducers see [24] and references cited therein.

[5] We also need to formulate a suitable probability model $\Pr(y \mid w; \theta)$ that can likewise be expressed as a weighted finite state transducer. It is clear that HMMs, CMMs, MEMMs and related models have this property.

Assume that there is a transducer M_θ over alphabets Γ and Σ with behavior $[M_\theta](y, w) = \Pr(y \,|\, w; \theta)$. The composition of T_{nd} with M_θ then has the following behavior, as desired: $[T_{\text{nd}} \circ M_\theta](x, w) = \sum_y tp(x, y) \ \Pr(y \mid w; \theta)$.

The next issue is to show how this calculation can be done for fixed $\langle x, w \rangle$. We write $\text{Str}(x)$ to denote a transducer[6] that maps the string pair $\langle x, x \rangle$ to 1 and all other pairs to 0. In order to evaluate (9) for fixed $\langle x, w \rangle$, construct the transducer

$$\text{Str}(x) \circ T_{\text{nd}} \circ M_\theta \circ \text{Str}(w), \tag{10}$$

which has the property that all paths leaving its start state are labeled with $\langle x, w \rangle$. Its behavior can be computed efficiently by a single-source algebraic path algorithm on the acyclic transition graph of the transducer (10) (see [25], §25.4).

We can simplify the construction of (10). When we build $\text{Str}(x) \circ T_{\text{nd}}$, we do not actually care about the first tape of the composed transducer and can eliminate it by projection/marginalization. Notice that the second projection $\pi_2(\text{Str}(x) \circ T_{\text{nd}})$ does not generally result in a deterministic automaton. However, we can directly construct a deterministic automaton that is equivalent to $\pi_2(\text{Str}(x) \circ T_{\text{nd}})$. The reason for doing so is to obtain a simple upper bound on the state and arc complexity of a transducer that carries out essentially the same computation as $\text{Str}(x) \circ T_{\text{nd}}$.

We construct a complete and unambiguous transducer T_{ua} that is equivalent to T_{nd}, meaning for each pair of same-length strings $\langle x, y \rangle$ there is precisely one accepting path through T_{ua}. Furthermore, the composition $\text{Str}(x) \circ T_{\text{ua}}$ is an output-deterministic transducer: its second projection is a deterministic automaton. Note that it is possible to have an unambiguous transducer for counting matching chunks in string pairs of a known length, but disambiguation of T_{nd} is impossible because T_{nd} can be used to count matching chunks in strings of unbounded length. In fact, the path multiplicity of T_{nd} is at the core of its design as a counter. However, an unambiguous version of T_{nd} can be constructed if we allow the set of states to be countably infinite. The initial portion of T_{ua} is shown in Fig. 2 (all edge weights are one; final weights are as indicated). In order to understand the correctness of T_{ua} (Fig. 2), observe its similarity to T_{nd} (Fig. 1). In both cases there are B:B transitions from the start state to a state with an I:I loop, as well as O:O, O:B and B:O transitions out of that state. However, whereas the B:B transition out of the second state of Fig. 1 signals the end of a chunk, it simultaneously signals the beginning of a new chunk (this is precisely the rationale for the B label – to encode adjacent junks), hence the vertical B:B transitions in Fig. 2. The final weight of each state corresponds to the number of matching chunks encountered. The states there are organized in two columns, with those on the left indicating that the inside of a matching chunk is being processed. The only way to get to the left column is to take a diagonal B:B transition that signals the beginning of a potential matching chunk. The two ways to proceed from the left column to the right column are to take an upward diagonal

[6] The construction of $\text{Str}(x)$ is a special case of the prefix tree (a. k. a. "trie") representation of a finite dictionary.

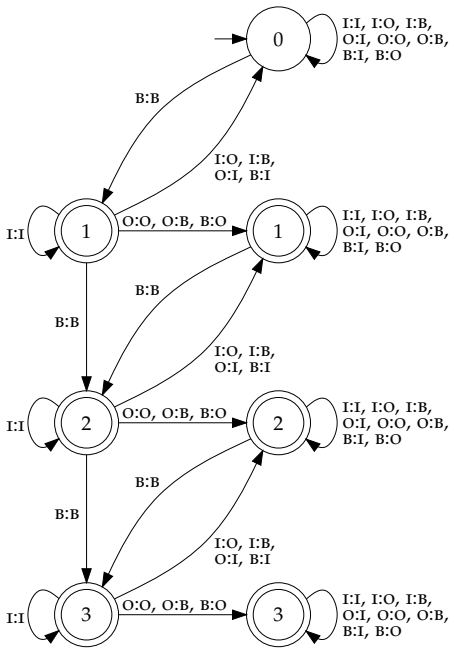

Fig. 2. Initial portion of T_{ua}.

transition, which indicates a failed potential match, or a horizontal transition, which successfully completes a true positive match.

In an implementation of finite state machines based on lazy data structures (e.g. [10]), infinite transducers like T_{ua} can be represented directly. For simplicity we will present a more traditional algorithm, shown in Fig. 3 (a+b), which constructs a deterministic finite automaton $T_{\text{det}}(x) = \pi_2(\text{Str}(x) \circ T_{\text{ua}})$. The states of $T_{\text{det}}(x)$ can be thought of as triples $\langle k, o, t \rangle$ where k is an index into x; o is a boolean variable that indicates if the state is part of a matching chunk; and t is the number of matching chunks encountered so far (equal to the final weights). Conceptually, k is a state of $\text{Str}(x)$ and $\langle o, t \rangle$ is a state of T_{ua}, where o selects the left ($o = \bot$) or right ($o = \top$) column of states in Fig. 2. There are at most $(|x| + 1) \times (2\,|x| + 1)$ states and a constant number of outgoing edges per state.

This is sufficient to guarantee that (10) can be evaluated efficiently. $M_\theta \circ \text{Str}(w)$ has $\Theta(3^j |w|)$ states when M_θ is a jth-order Markov model. Since j is fixed and x and w are of the same length and thus do not require separate indices, the composed automaton (10) has $\text{O}(|x|^2)$ states. Because the algebraic path computation runs in linear time, the overall computation of the expected number of true positives (9) runs in quadratic time. Moreover the hidden constant of proportionality is small when j is small and when there are few B labels in a label sequence, as is typically the case.

Tdet(x):

1: $\langle x_0, \ldots, x_{\ell-1} \rangle \leftarrow x$
2: $S \leftarrow \{\}$ // set of states
3: $F \leftarrow \{\}$ // set of final states
4: $E \leftarrow \{\}$ // set of transitions
5: $Q \leftarrow$ **new** Queue() // an empty queue
6: Q.enqueue($\langle 0, \top, 0, 0 \rangle$) // push start state
7: **while** $\neg\, Q$.isEmpty() **do**
8: $q \leftarrow Q$.dequeue()
9: **if** $q \in S$ **then**
10: **continue** // already visited q
11: $S \leftarrow S \cup \{q\}$
12: $k \leftarrow q[0]$ // index
13: **if** $k = \ell$ **then**
14: $F \leftarrow F \cup \{q\}$
15: **continue** // final state
16: *outside* $\leftarrow q[1]$ // outside a match?
17: $tp \leftarrow q[2]$ // num. matching chunks so far
18: **if** $x_k = \text{B} \vee x_k = \text{O} \vee (\mathit{outside} \wedge x_k = \text{I})$ **then**
19: addEdge($E, Q, q,$ O, $\top, 0$)
20: **if** $x_k = \text{B}$ **then**
21: addEdge($E, Q, q,$ B, $\bot, +1$)
22: **else**
23: addEdge($E, Q, q,$ B, $\top, 0$)
24: **if** *outside* $= \top$ **then**
25: addEdge($E, Q, q,$ I, $\top, 0$)
26: **else if** $x_k = \text{I}$ **then**
27: **assert** $tp > 0$
28: addEdge($E, Q, q,$ I, $\bot, 0$)
29: addEdge($E, Q, q,$ O, $\top, -1$)
30: addEdge($E, Q, q,$ B, $\top, -1$)
31: **else**
32: **assert** $tp > 0$
33: addEdge($E, Q, q,$ I, $\top, -1$)
34: **return** $\langle S, \langle 0, \top, 0, 0 \rangle, F, E \rangle$

(a) Construction of T_{det}.

addEdge($E, Q, q, \mathit{osym}, \mathit{outside}, \Delta tp$):

1: $k \leftarrow q[0] + 1$
2: $tp \leftarrow q[2] + \Delta tp$
3: $r \leftarrow \langle k, \mathit{outside}, tp, 0 \rangle$ // target
4: $E \leftarrow E \cup \{\langle q, \mathit{osym}, r \rangle\}$
5: $Q \leftarrow Q \cup \{r\}$

(b) Computing exp. true positives.

addEdge($E, Q, q, \mathit{osym}, \mathit{outside}, \Delta tp$):

1: $k \leftarrow q[0] + 1$
2: $tp \leftarrow q[2] + \Delta tp$
3: **if** $\mathit{osym} = \text{B}$ **then**
4: $pm \leftarrow q[3] + 1$
5: **else**
6: $pm \leftarrow q[3]$
7: $r \leftarrow \langle k, \mathit{outside}, tp, pm \rangle$ // target
8: $E \leftarrow E \cup \{\langle q, \mathit{osym}, r \rangle\}$
9: $Q \leftarrow Q \cup \{r\}$

(c) Computing expected utility.

U(x, w, θ, β):

1: $mx \leftarrow$ num. occurrences of B in x
2: **if** $mx = 0$ **then**
3: **return** $\Pr(x \mid w; \theta)$
4: $T \leftarrow \text{Tdet}(x)$
5: $\langle D, F \rangle \leftarrow \text{algPathCompose}(T, \theta)$
6: $u \leftarrow 0$
7: **for each** $q \in F$ **do**
8: $tp \leftarrow q[2]$
9: $my \leftarrow q[3]$
10: $u \leftarrow u + D[q]\ tp/(mx + \beta\, my)$
11: **return** $(\beta + 1)\, u$

(d) Overall computation.

Fig. 3. Algorithm for constructing T_{det}.

3.2 Computing Expected Utility

Evaluating expected F_β also involves computing the expected number of hypothesized chunks $\sum_y m(y)\ \Pr(y \mid w; \theta)$. This is straightforward: whenever one encounters the label B, one increments a counter. An FST which counts (up to a fixed threshold of n) the chunks it sees on its second tape is shown in Fig. 4.

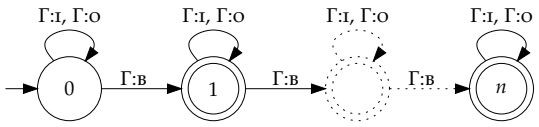

Fig. 4. Output-deterministic transducer that counts hypothesized chunks.

An FST that computes $\langle tp(x,y), m(y)\rangle$ in parallel can be obtained[7] by a composition-like combination of the transducers in Fig. 2 and Fig. 4. This is quite simple: the algorithm in Fig. 3 (a+b) can be extended by making states quadruples instead of triples, changing only the function "addEdge", so that the number of hypothesized chunks can be tracked directly. The extended algorithm in Fig. 3 (a+c) constructs a transducer with at most $(|x|+1)^2 \times (2\,m(x)+1)$ states, whose behavior can therefore be computed in cubic time. The corresponding algorithm is shown in Fig. 3 (a+c+d) and requires a subroutine (not shown for reasons of space) that computes (i) the composition of T_{det} with the automaton $M_\theta \circ \text{Str}(y)$ representing the probability model and (ii) the algebraic path weights for the final states of the composed transducer. The key insight here is that these final states partition the probability mass of the probability model in such a way that all label strings with the same number of true positive matches and the same number of predicted chunks contribute to just one final state. Whereas there are exponentially many label sequences, there are only quadratically many final states. Therefore the expected utility $\mathcal{U}$ can be evaluated efficiently in polynomial time.

4 Conclusion

We have presented an algorithm for efficiently computing the expected utility of hypotheses produced by a stochastic chunker within the framework of weighted transducers and automata. This has direct applications in loss-sensitive training of stochastic chunker models, and in decoding procedures that seek to maximize the F-measure. The key insight is that the number of matching chunks in two label sequences can be counted efficiently using an unambiguous infinite state transducer. This does not transcend the boundaries of finite state computations, since only finite prefixes and finitely many hypotheses are considered at all times. Expressing the machine for counting matching chunks as an infinite state transducer enabled us to state simple bounds on the size of derived machines. The chunk counting transducer was extended trivially to also keep track of predicted chunks, thus computing matching and predicted chunks in parallel, as required for the computation of expected utility.

[7] More precisely, since this computation involves pairs of real numbers, the weights of the component transducers must be thought of as having been mapped into a direct product of the real semiring with itself. Composition takes place in that product semiring. The final weights are tuples $\langle tp, pm\rangle$.

References

1. Church, K.W.: A stochastic parts program and noun phrase parser for unrestricted text. In: ANLP. (1988) 136–143
2. Voutilainen, A.: NPtool, a detector of English noun phrases. In: WVLC. (1993) 48–57
3. Ramshaw, L.A., Marcus, M.P.: Text chunking using transformation-based learning. In: WVLC. (1995) 82–94
4. Tjong Kim Sang, E.F., Veenstra, J.: Representing text chunks. In: EACL. (1999) 173–179
5. Punyakanok, V., Roth, D.: The use of classifiers in sequential inference. In: NIPS. (2000) 995–1001
6. Bikel, D.M., Miller, S., Schwartz, R., Weischedel, R.: Nymble: A high-performance learning name-finder. In: ANLP. (1997) 194–201
7. Freitag, D.: Toward general-purpose learning for information extraction. In: COLING-ACL. (1998) 404–408
8. Zhou, G.: Chunking-based Chinese word tokenization. In: SIGHAN. (2003)
9. van Rijsbergen, C.J.: Foundation of evaluation. Journal of Documentation **30** (1974) 365–373
10. Mohri, M., Pereira, F., Riley, M.: The design principles of a weighted finite-state transducer library. Theoretical Computer Science **231** (2000) 17–32
11. Bengio, Y.: Markovian models for sequential data. Neural Computing Surveys **2** (1999) 129–162
12. Dietterich, T.G.: Machine learning for sequential data: A review. Lecture Notes in Computer Science **2396** (2002)
13. Collins, M.: Machine learning methods in natural language processing. Tutorial presented at COLT (2003)
14. Ratnaparkhi, A.: Maximum Entropy Models for Natural Language Ambiguity Resolution. PhD thesis, University of Pennsylvania (1998)
15. Tjong Kim Sang, E.F.: Introduction to the CoNLL-2002 shared task. In: CoNLL. (2002) 155–158
16. Duda, R.O., Hart, P.E., Stork, D.G.: Pattern Classification. 2nd edn. Wiley (2000)
17. Zhang, T., Damerau, F., Johnson, D.: Text chunking using regularized winnow. In: ACL. (2001) 539–546
18. Zhang, T., Damerau, F., Johnson, D.: Text chunking based on a generalization of winnow. Journal of Machine Learning Research **2** (2002) 615–637
19. Zhang, T., Johnson, D.: A robust risk minimization based named entity recognition system. In: CoNLL. (2003) 204–207
20. Stolcke, A., König, Y., Weintraub, M.: Explicit word error minimization in n-best list rescoring. In: EuroSpeech. (1997)
21. Kumar, S., Byrne, W.: Minimum Bayes-risk decoding for machine translation. In: HLT-NAACL. (2004) 169–176
22. van Rijsbergen, C.J.: Information Retrieval. 1st edn. Butterworths (1975)
23. Mohri, M., Pereira, F., Riley, M.: Weighted automata in text and speech processing. In: ECAI'96 Workshop on Extended Finite State Models of Language. (1996) 46–50
24. Allauzen, C., Mohri, M., Roark, B.: Generalized algorithms for constructing language models. In: ACL. (2003) 40–47
25. Cormen, T.H., Leiserson, C.E., Rivest, R.L.: Introduction to Algorithms. 1st edn. MIT Press (1990)