Lexical Event Ordering with an Edge-Factored Model

Citation for published version:
<http://www.aclweb.org/anthology/N/N15/N15-1122.pdf>

Link:
Link to publication record in Edinburgh Research Explorer

Document Version:
Peer reviewed version

Published In:
Proceedings of NAACL 2015

General rights
Copyright for the publications made accessible via the Edinburgh Research Explorer is retained by the author(s) and / or other copyright owners and it is a condition of accessing these publications that users recognise and abide by the legal requirements associated with these rights.

Take down policy
The University of Edinburgh has made every reasonable effort to ensure that Edinburgh Research Explorer content complies with UK legislation. If you believe that the public display of this file breaches copyright please contact openaccess@ed.ac.uk providing details, and we will remove access to the work immediately and investigate your claim.
Abstract

Extensive lexical knowledge is necessary for temporal analysis and planning tasks. We address in this paper a lexical setting that allows for the straightforward incorporation of rich features and structural constraints. We explore a lexical event ordering task, namely determining the likely temporal order of events based solely on the identity of their predicates and arguments. We propose an "edge-factored" model for the task that decomposes over the edges of the event graph. We learn it using the structured perceptron. As lexical tasks require large amounts of text, we do not attempt manual annotation and instead use the textual order of events in a domain where this order is aligned with their temporal order, namely cooking recipes.

1 Introduction

Temporal relations between events are often implicit, and inferring them relies on lexical and world knowledge about the likely order of events. For instance, to execute the instruction "fry the onion," the hearer should probably obtain oil beforehand, even if not instructed so explicitly. Lexical knowledge about the likely order of events is therefore necessary for any semantic task that requires temporal reasoning or planning, such as classifying temporal relations (Mani et al., 2006; Lapata and Lascarides, 2006; Yoshikawa et al., 2009; D’Souza and Ng, 2013; Mirza and Tonelli, 2014, inter alia), textual entailment (Dagan et al., 2013) or temporal information extraction (Ling and Weld, 2010). Lexical temporal knowledge is further important for modeling grammatical phenomena such as tense and aspect (Steedman, 2002).

In this paper we address the task of lexical event ordering, namely predicting the ordering of events based only on the identity of the words comprising their predicates and arguments. Concretely, the task is to predict the order of an unordered set of predicate-argument structures. Predicting the likely order of event types is a step towards more intricate planning and reasoning scenarios (see §3), and is useful in itself for tasks such as concept-to-text generation (Reiter et al., 2000), or in validating the correctness of instruction sets. A related idea can be found in modeling sentence coherence (Lapata, 2003; Barzilay and Lapata, 2008, inter alia), although here we focus on lexical relations between events, rather than coherence relations between complete sentences.

Compiling a resource of temporal tendencies between events can hardly be done manually, given the number and wealth of phenomena that have to be accounted for. Temporally annotated corpora, often annotated according to TimeML principles (Pustejovsky et al., 2003), are a useful resource for studying temporal relations. However, due to incurred costs, annotated corpora are too small for most lexical tasks. For instance, the TimeML annotated data used in the latest TempEval shared task contains only 100K words or so (UzZaman et al., 2013).

Previous work that does not rely on manually annotated data has had some success in discovering temporal lexical relations between predicates (Chklovski and Pantel, 2004; Chambers and Jurafsky, 2008b; Talukdar et al., 2012). However, despite their appeal, these methods have mostly fo-
cused on inducing simple event types, consisting of single words (e.g., “buy-own”) or fixed expressions, and are hard to extend to include rich features (e.g., order-based and pattern-based features). Furthermore, measuring recall without annotated data is notoriously difficult, and evaluation is often precision-based or extrinsic.

We take a graph-based structured prediction approach to the task, motivated by the flexibility it allows in incorporating various feature sets and constraints. We use an edge-factored model, which decomposes over the edges in the graph of events comprising the recipe (§4). We estimate the model using the structured perceptron algorithm. We compare the structured perceptron approach to an approximate greedy baseline and to a locally normalized model reminiscent of common approaches for order learning, obtaining superior results (§8). The learning algorithm is of potential use in other ordering tasks such as machine translation reordering (Tromble and Eisner, 2009).

We focus on domains in which the order of events in the text is aligned with their temporal order. By doing so we avoid the costly and error-prone manual annotation of temporal relations by using the textual order of recipes to approximate their temporal order.1 Specifically, we address the cooking recipes domain, which we motivate in §2.

In summary, the contribution of this paper is three-fold: (1) we explore the task of lexical event ordering and means for its evaluation; (2) we present an edge-factored model for the task, and show it can be used to predict the order of events well (77.7% according to standard measures for ordering evaluation); (3) we present a method for extracting events and create a dataset of ordered events using recipes extracted from the web.

2 Related Work

Temporal semantics is receiving increasing attention in recent years. Lexical features are in frequent use and rely in most part on external resources which are either manually compiled or automatically induced. The line of work most closely related to ours focuses on inducing lexical relations between event types. Most work has been unsupervised, often using pattern-based approaches relying on manually crafted (Chklovski and Pantel, 2004) or induced patterns (Davidov et al., 2007), that correlate with temporal relations (e.g., temporal discourse connectives). Talukdar et al. (2012) uses the textual order of events in Wikipedia biographical articles to induce lexical information. We use both textual order and discourse connectives to define our feature set, and explore a setting which allows for the straightforward incorporation of additional features.

Chambers and Jurafsky (2008b; 2009) addressed the unsupervised induction of partially ordered event chains (or schema) in the news domain, centered around a common protagonist. One of their evaluation scenarios tackles a binary classification related to event ordering, and seeks to distinguish ordered sets of events from randomly permuted ones, yielding an accuracy of 75%. Manshadi et al. (2008) used language models to learn event sequences and conducted a similar evaluation on weblogs with about 65% accuracy. The classification task we explore here is considerably more complex (see §8).

The task of script knowledge induction has been frequently addressed in recent years. Balasubramanian et al. (2013) and Pichotta and Mooney (2014) extended Chambers and Jurafsky’s model to include events that have multiple arguments. Jans et al. (2012) use skip-grams to capture event-event relations between not necessarily consecutive events. Regneri et al. (2010) constructed a temporal lexical knowledge base through crowd-sourcing. Their approach is appealing as it greatly reduces the costs incurred by manual annotation and can potentially be used in conjunction with lexical information obtained from raw text. Modi and Titov (2014) jointly learn the stereotypical order of events and their distributional representation, in order to capture paraphrased instances of the same event type. Frermann et al. (2014) models the joint task of inducing event paraphrases and their order using a Bayesian framework. All latter three works evaluated their induced temporal ordering knowledge on a binary prediction of whether a temporal relation between a pair of (not necessarily related) events holds, and not on the prediction of a complete permutation given an unordered event set as in this work. Their evaluation was conducted on 30 event pairs manually an-

1See Cassidy et al. (2014) for a discussion of inter-annotator agreement in TimeML-based schemes.
notated through crowd-sourcing, where Modi and Titov (2014) further included an evaluation on a large set of pairs automatically extracted from the Gigaword corpus.

The appeal of the cooking domain for studying various semantic phenomena has been recognized by several studies in NLP and AI (Tasse and Smith, 2008; Bollini et al., 2013; Cimiano et al., 2013; Regneri et al., 2013; Malmaud et al., 2014). The domain is here motivated by several considerations. First, recipes mostly describe concrete actions, rather than abstract relations, which are less relevant to temporal ordering. Second, from a practical point of view, many recipes are available online in computer-readable format. Third, the restrictiveness of the cooking domain can also be seen as an advantage, as it can reveal major conceptual challenges raised by the task, without introducing additional confounds.

3 Temporally Ordering Lexical Events

We formalize our task as follows. Let \( U \) be a set of event types, namely actions or states (represented as predicates) and objects which these actions operate on (represented as arguments to the predicates; mostly ingredients or kitchenware). Formally, each \( e \in U \) is a tuple \( \langle a, c_1, \ldots, c_n \rangle \) where \( a \) is the main verb or predicate describing the event (such as “stir” or “mix”) and \( c_1, \ldots, c_n \) is a list of arguments that the predicate takes (e.g., “salt” or “spoon”). Two additional marked events, \( s \) and \( f \), correspond to “start” and “finish” events. A recipe is a sequence of events in \( U \), starting at \( s \) and ending at \( f \).

Given a recipe \( R = \langle e_1, \ldots, e_m \rangle \), we wish to predict the order of the events just from the (multiset) \( \{e_i\}_{i=1}^m \). In this work we use the textual order of events to approximate their temporal order (see, e.g., Talukdar et al. (2012) for a similar assumption). The validity of this assumption for cooking recipes is supported in §6.

Figure 1 gives an example of a set of events extracted from our dataset for the dish “Apple Crisp Ala [sic] Brigitte.” Lexical information places quite a few limitations on the order of this recipe. For instance, in most cases serving is carried out at the end while putting the ingredients in is done prior to baking them. However, lexical knowledge in itself is unlikely to predict the exact ordering of the events as given in the recipe (e.g., spreading butter might be done before or after baking).

One of the major obstacles in tackling planning problems in AI is the knowledge bottleneck. Lexical event ordering is therefore a step towards more ambitious goals in planning. For instance, temporal relations may be used to induce planning operators (Mourão et al., 2012), which can in turn be used to generate a plan (recipe) given a specified goal and an initial set of ingredients.

4 Model, Inference and Learning

In this section we describe the main learning components that compose our approach to event ordering.

4.1 Edge-Factored Model for Event Ordering

We hereby detail the linear model we use for ordering events. Let \( S = \{e_1, \ldots, e_m\} \subseteq U \) be a set of events as mentioned in §3. Let \( G(S) = (S \cup \{s, f\}, E(S)) \) be an almost-complete directed graph with \( E(S) = (S \cup \{s\}) \times (S \cup \{f\}) \subseteq U \times U \). Every Hamiltonian path\(^2\) in \( G(S) \) that starts in \( s \) and ends in \( f \) defines an ordering of the events in \( S \). The edge \( (e_i, e_j) \) in such a path denotes that \( e_i \) is the event that comes before \( e_j \).

The modeling problem is to score Hamiltonian paths in a given directed graph \( G(S) \). Here, we use an edge-factored model. Let \( \phi: (U \times U) \rightarrow \mathbb{R}^d \) be a feature function for pairs of events, represented as directed edges. In addition, let \( \theta \in \mathbb{R}^d \) be a weight vector. We define the score of a Hamiltonian path \( h = (h_1, \ldots, h_{m+1}) \) (\( h_i \in E(S) \)) as:

\[
\text{score}(h|S) = \sum_{i=1}^{m+1} \theta^\top \phi(h_i) \tag{1}
\]

Given a weight vector \( \theta \) and a set of events \( S \), inference is carried out by computing the highest scoring Hamiltonian path in \( G(S) \):

\[
h^* = \arg \max_{h \in H(S)} \text{score}(h|S) \tag{2}
\]

where \( H(S) \) is the set of Hamiltonian paths in \( G(S) \) that start with \( s \) and end with \( f \). The path \( h^* \) is the best temporal ordering of the set of events \( S \) according to the model in Eq. 1 with weight vector \( \theta \).

\(^2\)A path in a graph that visits all nodes exactly once.
4.2 Inference

As mentioned above, inference with the edge-
factored model requires solving the maximization
problem in Eq. 2. This corresponds to finding a
Hamiltonian path in a complete graph, which is
generally an NP-hard problem. Reasonable approxima-
tions for this problem are also NP-hard. Still tech-
niques are developed for specialized cases, due to
the problem’s importance in discrete optimization.

Despite its theoretical NP-hardness, this maxi-
mization problem can be represented as an Integer
Linear Program (ILP), and then solved using generic
techniques for ILP optimization. Due to the rela-
tively short length of recipes (13.8 events on average
in our corpus), the problem can be effectively solved
in most cases.

The proposed algorithmic setting is appealing for
its flexibility. The linear score formulation allows
us to use rich features, while using ILP allows to
easily incorporate structural constraints. Indeed, ILP
has been proven valuable in various NLP tasks (Roth
and Yih, 2007; Talukdar et al., 2012; Scaria et al.,
2013). See Appendix A for our ILP formulation.

As a baseline, we experiment with an additional
greedy inference algorithm, similar to the one
described by Lapata (2003) for sentence ordering. The
algorithm iteratively selects an outgoing edge (start-
ing from the node $s$) that has the largest weight to
a node that has not been visited so far, until all ver-
tices are covered, at which point the path terminates
by traveling to $f$.

4.3 Learning

The learning problem takes as input a dataset con-
isting of unordered sets of events, paired with a tar-
get ordering. We consider two types of learning al-
gorithms for the edge-factored model in the previous
section. The first learns in a global training setting
using the averaged structured perceptron (Collins,
2002), with the decoding algorithm being either the
one based on ILP (henceforth, GLOBAL-PRC), or
the greedy one (GREEDY-PRC). Given a training in-
stance $S$ and its correct label $h^c$, the structured
perceptron calls the inference procedure as a subroutine
and updates the weight vector $\theta$ according to the
difference between the value of the feature function on
the predicted path ($\sum_{h^*} \phi(h_i)$) and on the correct
path ($\sum_{h^c} \phi(h_i)$).

The second learning algorithm we try is based on
factored training. This algorithm maximizes the
likelihood of a conditional log-linear model $p$:

$$
p(e_2|e_1, \theta, S) = \frac{\exp(\theta^T \phi(e_1, e_2))}{Z(\theta, S, e_1)}
$$

$$
Z(\theta, S, e_1) = \sum_{e: (e_1, e) \in E(S)} \exp(\theta^T \phi(e_1, e))
$$

where $e_1, e_2 \in S \cup \{s, f\}$. This is a locally norm-
alyzed log-linear model that gives the probability of
transitioning to node $e_2$ from node $e_1$. Maximizing
the score in Eq. 1 has an interpretation of finding the
highest scoring path according to an edge-factored
Markovian model, such that:
where \( h = (h_1, \ldots, h_{m+1}) \) is a Hamiltonian path with \( h_i = (e_{i-1}, e_i) \) being a directed edge in the path. Initial experimentation suggested that greedy inference (henceforth, \textsc{greedy-loglin}) works better in practice than the ILP formulation for the locally-normalized model. We therefore do not report results on global inference with this log-linear model. We suspect that greedy inference works better with the log-linear model because it is trained locally, while the perceptron algorithm includes a global inference step in its training, and therefore better matches global decoding.

\textsc{greedy-loglin} closely resembles the learning model of Lapata (2003), as both are first-order Markovian and use the same (greedy) inference procedure. Lapata’s model differs from \textsc{greedy-loglin} in being a generative model, where each event is a tuple of features, and the transition probability between events is defined as the product of transition probabilities between feature pairs. \textsc{greedy-loglin} is discriminative, so to be maximally comparable to the presented model.

5 The Feature Set

Table 1 presents the complete set of features. We consider three sets of features: Lexical encodes the written forms of the event pair predicates and objects; \textsc{Brown} uses Brown clusters (Brown et al., 1992) to encode similar information, but allows generalization between distributionally similar words; and Frequency encodes the empirical distribution of temporally-related phenomena.

The feature definitions make use of several functions. For brevity, we sometimes say that an event \( e \) is \((a, c_1)\) if \( e \)’s predicate is \( a \) and its first argument is \( c_1 \), disregarding its other arguments. Let \( C \) be a reference corpus of recipes for collecting statistics. The function \( B(w) \) gives the Brown cluster of a word \( w \), as determined by clustering \( C \) into 50 clusters \( \{1, \ldots, 50\} \). The function \( \text{ORD}(a, c) \) returns the mean ordinal number of an \((a, c)\) event in \( C \). The ordinal number of the event \( e_i \) in a recipe \( (e_1, \ldots, e_m) \) is defined as \( i - \frac{m}{2} \).

<table>
<thead>
<tr>
<th>Template</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>((a^+, a^-))</td>
<td>(\text{fry}, \text{add})</td>
</tr>
<tr>
<td>((a^+, c_1^-))</td>
<td>(\text{fry}, \text{oil})</td>
</tr>
<tr>
<td>((a^+, c_1^+))</td>
<td>(\text{onions}, \text{add})</td>
</tr>
<tr>
<td>((c_1^-, c_2^-))</td>
<td>(\text{onions}, \text{oil})</td>
</tr>
</tbody>
</table>

\[ p(h|\theta, S) = \prod_{i=1}^{m+1} p(e_i|e_{i-1}, \theta, S), \]

\( B(a^+) \), \( B(c_1^-) \), and \( B(c_1^+) \) are Brown clusters (Brown et al., 1992) to encode similar information, but allows generalization between distributionally similar words. \( \forall \ell \in L, \ell + P(\theta) \) is a binary feature indicating the appearance of an event with \( a^+ = \text{fry} \) on one end of the edge and an event with \( a^- = \text{add} \) on the other end of it.

\[ \epsilon = 10^{-3} \] in our experiments. See text for elaboration.

We further encode the tendency of two events to appear with temporal discourse connectives, such as “before” or “until.” We define a linkage between two events as a triplet \((e_1, e_2, \ell) \in (U \times U \times L)\), where \( L \) is the set of linkage types, defined according to their marker’s written form. §6 details the extraction process of linkages from recipes. We further include a special linkage type \textsc{linear} based on the order of events in the text, and consider every pair of events \( e_1 \) and \( e_2 \) that follow one another in a recipe as linked under this linkage type.

For each linkage type \( \ell \in L \), we define an empirical probability distribution \( P_\ell((a, c_1), (a', c'_1)) = \frac{p((a, c_1), (a', c'_1)|\ell)}{p((a, c_1), (a', c'_1)),} \) based on simple counting. The function PMI gives the point-wise mutual information of two events and is defined as:

\[ \text{PMI}((a, c), (a', c')) = \log \left( \frac{P_\ell((a, c_1), (a', c'_1))}{P_\ell((a, c_1)) \cdot P_\ell((a', c'_1))} \right) \]

Frequency-based features encode the empirical estimate of the probabilities that various pairs of features would occur one after the other or linked with a discourse marker. They are equivalent to using probabilities extracted from maximum likelihood estima-
tion according to a bigram model in the discriminative learning. While some of this information is implicitly found in the lexical features, collecting frequency counts from a large training set is much quicker than running costly structured optimization. Rather the discriminative training can weigh the different empirical probabilities according to their discriminative power. Indeed we find that these features are important in practice and can result in high accuracy even after training on a small training set.

6 The Recipe Dataset

Data and Preprocessing. The data is extracted from a recipe repository found on the web. The recipes are given as free text. To extract event types we run the Stanford CoreNLP pipeline of a tokenizer, POS tagger, a lexical constituency parser (the EnglishPCFG parsing model) and extract typed Stanford dependencies (de Marneffe and Manning, 2008). As is common with web extractions, the recipes contain occasional spelling, grammatical and formatting errors. The corpus consists of 139 files, 73484 recipes, 1.02M events (13.8 events per recipe on average) and 11.05M words.

Event Extraction. We focus on verbal events and do not extract nominal and adjectival argument structures, which are not as well supported by current parsing technology. Any verb is taken to define an event, aside from modal verbs, auxiliaries and secondary verbs. A secondary verb (e.g., “let,” “begin”) does not describe an action in its own right, but rather modifies an event introduced by another verb. We identify these verbs heuristically using a list given in Dixon (2005, p. 490–491) and a few simple rules defined over parse trees. E.g., from the sentence “you should begin to chop the onion,” we extract a single event with a predicate “chop.” Arguments are taken to be the immediate dependents of the predicate that have an argument dependency type (such as direct or indirect objects) according to the extracted Stanford dependencies. For prepositional phrases, we include the preposition as part of the argument. Argument indices are determined by their order in the text. The order of events is taken to be the order of their verbs in the text.

Linkage Extraction. We focus on a subset of linkage relations, which are relevant for temporal relations. We use Pitler and Nenkova’s (2009) explicit discourse connectives classifier to identify temporal discourse linkers, discarding all other discourse linkers. Once a discourse linker has been detected, we heuristically extract its arguments (namely the pair of verbs it links) according to a deterministic extraction rule defined over the parse tree. We find 28 distinct connectives in our training set, where the 5 most common linkers “until,” “then,” “before,” “when” and “as” cover over 95% of the instances. We extract 36756 such linkages from the corpus, 0.5 linkages per recipe on average.

Temporal and Textual Ordering. In order to confirm that temporal and textual order of recipes are generally in agreement, we manually examine the first 20 recipes in our development set. One recipe was excluded as noise, resulting in 19 recipes and 353 events. We identify the sources of misalignment between the linear order and the temporal order of the events. 13 events (3.7%) did not have any clear temporal orderings. These consisted of mostly negations and modalities (e.g., “do not overbrown!”), sub-section headings (e.g., “Preparation”) or other general statements that do not constitute actions or states. For the remaining 293 events, we compare their linear and the temporal orderings. We estimate the frequency of sub-sequences that contradict the temporal order and confirm that they occur only infrequently. We find that most disagreements fall into these two categories: (1) disjunctions between several events, only one of which will actually take place (e.g., “roll Springerle pin over dough, or press mold into top”); (2) a pair, or less commonly a triplet, of events are expressed in reverse order. For instance, “place on greased and floured cookie sheet,” where greasing and flouring should occur before the placing action. We note that assuming the alignment of the temporal and textual order

---

3 http://www.ffts.com/recipes.htm
4 http://nlp.stanford.edu/software/corenlp.shtml
5 Links to the original recipes, the preprocessed recipes and all extracted events can be found in http://homepages.inf.ed.ac.uk/oabend/event_order.html.

6 This did not result from an extraction problem, but rather from the recipe text itself being too noisy to interpret.

7 Events are parsed manually so to avoid confounding the results with the parser’s performance.
of recipes does not suggest that the textual order is the only order of events that would yield the same outcome.

We compute the Kendall’s Tau correlation, a standard measure for information ordering (Lapata, 2006), between the temporal and linear orderings for each recipe. In cases of several events that happen simultaneously (including disjunctions), we take their ordinals to be equal. For instance, for three events where the last two happen at the same time, we take their ordering to be (1,2,2) in our analysis. We find that indeed temporal and textual orderings are in very high agreement, with 6 recipes of the 19 perfectly aligned. The average Kendall’s Tau between the temporal ordering and the linear one is 0.924.

7 Experimental Setup

Evaluation. We compute the accuracy of our algorithms by comparing the predicted order to the one in which the events are written. We first compute the number of exact matches, denoted with EXACT, namely the percentage of recipes in which the predicted and the textual orders are the same.

For a more detailed analysis of imperfect predictions, we compute the agreement between sub-sequences of the orderings. We borrow the notion of a “concordant pair” from the definition of Kendall’s Tau and generalize it to capture agreement of longer sub-sequences. Two $k$-tuples of integers $(x_1, ..., x_k)$ and $(y_1, ..., y_k)$ are said to “agree in order” if for every $1 \leq i < j \leq k$, $x_i < x_j$ iff $y_i < y_j$. Given two orderings of the same recipe $O_1 = (e_{\tau(1)}, ..., e_{\tau(m)})$ and $O_2 = (e_{\sigma(1)}, ..., e_{\sigma(m)})$ (where $\tau$ and $\sigma$ are permutations over $[m] = \{1, ..., m\}$) and given a sequence of $k$ monotonically increasing indices $t = (i_1, ..., i_k)$, $t$ is said to be a “concordant $k$-tuple” of $O_1$ and $O_2$ if $\tau(i_1, ..., \tau(i_k))$ and $\sigma(i_1, ..., \sigma(i_k))$ agree in order, as defined above.

Denote the unordered recipes of the test data as $\{R_i\}_{i=1}^N$, where $R_i = \{e_{i1}^1, ..., e_{im_i}^i\} \subset U$ for all $i$, and their target orderings $\Sigma = \{\sigma_i\}_{i=1}^N$, where $\sigma_i$ is a permutation over $[m_i]$. Assume we wish to evaluate a set of predicted orderings for this test data $T = \{\tau_i\}_{i=1}^N$, where again $\tau_i$ is a permutation over $[m_i]$. Denote the number of concordant $k$-tuples of $\sigma_i$ and $\tau_i$ as $\text{conc}(\sigma_i, \tau_i)$. The total number of of monotonically increasing $k$-tuples of indices is $\binom{m_i}{k}$. The $k$-wise (micro-averaged) accuracy of $T$ with respect to $\Sigma$ is:

$$\text{acc}_k(\Sigma, T) = \frac{\sum_{i=1}^N \text{conc}(\sigma_i, \tau_i)}{\sum_{i=1}^N \binom{m_i}{k}}$$

Any $k$-tuples containing the start node $s$ or the end node $f$ are excluded, as their ordering is trivial. Recipes of length less than $k$ are discarded when computing $\text{acc}_k$. A micro-averaged accuracy measure is used so as not to disproportionately weigh short recipes. However, in order to allow comparison to mean Kendall’s Tau, commonly used in works on order learning, we further report a macro-averaged $\text{acc}_2$ by computing $\text{acc}_2$ for each recipe separately, and taking the average of resulting accuracy levels. Average Kendall’s Tau can now be computed by $2\text{acc}_2 - 1$ for the macro-averaged $\text{acc}_2$ score.

Data. We randomly partition the text into training, test and development sets, taking an 80-10-10 percent split. We do not partition the individual files so as to avoid statistical artifacts introduced by recipe duplications or near-duplications. The training, development and test sets contain 58038, 7667 and 7779 recipes respectively. The total number of feature template instantiations in the training data is 8.94M.

Baselines and Algorithms. We compare three learning algorithms. GLOBAL-PRC is the structured perceptron algorithm that uses ILP inference. GREEDY-PRC is a structured perceptron in which inference is done greedily. GREEDY-LOGLIN is the locally normalized log-linear model with greedy inference. RANDOM randomly (uniformly) selects a permutation of the recipe’s events.

Experimental Settings. The structured perceptron algorithms, GLOBAL-PRC and GREEDY-PRC, are run with a learning rate of 0.1 for 3 iterations. To avoid exceedingly long runs, we set a time limit in seconds $\beta$ on the running time of each ILP inference stage used in GLOBAL-PRC. We consider two training scenarios: 4K, which trains on the first 4K recipes of the training set, and 58K, which trains on the full training data of 58K recipes. In GLOBAL-PRC we set $\beta$ to be 30 seconds for the 4K
yields no further improvement.

Previous studies evaluated their models on the related problem of distinguishing randomly permuted and correctly ordered chains of events (§2). In this paper we generalize this task to complete event or-

ted problem of distinguishing randomly permuted


tion probabilities \( P(e_i|e_j) \) between events \( e_i \) and \( e_j \) (in addition to the regular transition probabilities \( P(e_i|e_j) \)). However, preliminary experimentation with this technique did not yield improved performance. Future work will address higher-order models that straightforwardly account for such long-distance dependencies.

To qualitatively assess what generalizations are learned by the model, we apply GLOBAL-PRC to the development data and look at what event pairs obtained either particularly high or particularly low results. For each pair of predicates and their first arguments \( (a^1,c^1) \), \( (a^2,c^2) \), we compute the average weight of an edge connecting events of these types, discarding pairs of frequency less than 20.

The 20 highest scoring edges contain pairs such as (“add,” “mixing after addition”), (“beat whites,” “fold into mixture”) and (“cover for minutes,” “cook”), in addition to a few noisy pairs resulting from parser errors. The 20 lowest scoring edges contain event pairs that are likely to appear in the opposite order. 11 of the cases include as a first argument the predicates “serve,” “cool” or “chill,” which are likely to occur at the end of a recipe. 3 other edges linked duplications (e.g., (“reduce heat,” “reduce heat”)), which are indeed unlikely to immediately follow one another. These findings suggest the importance of detecting both lexical pairs that are unlikely to follow one another, in addition to those that are likely to.

9 Conclusion

We addressed the problem of lexical event ordering, and developed an edge-factored model for tackling it. We rely on temporally aligned texts, using a new dataset of cooking recipes as a test case, thereby avoiding the need for costly and error-prone manual annotation. We present results of a pair-wise accuracy of over 70% using a basic set of features, and show the utility of the structured perceptron algorithm over simpler greedy and local approaches. The setup we explore, which uses a discriminative model and an ILP formulation, is easy to extend both in terms of features and in terms of more complex formal constraints and edge dependencies, as was done in graph-based dependency parsing (McDonald et al., 2005). Future work will address the extension of the feature set and model, and the application of this model to temporal semantics and planning tasks. We will further address the application of semi-supervised variants of the proposed techniques (e.g., self-training) to other domains, where no sizable corpora of temporally aligned data can be found.

Acknowledgments

We would like to thank Nathan Schneider, Roy Schwartz, Bonnie Webber and the members of the Probmodels group at the University of Edinburgh for helpful comments. This work was supported by ERC Advanced Fellowship 249520 GRAMPLUS.

Appendix A: Maximal Hamiltonian Path

Let \( G(S) = (S \cup \{s,f\}, E(S)) \) be an almost-complete directed graph with \( E = E(S) = (S \cup \{s\}) \times (S \cup \{f\}) \). Let \( c_{ij} \in \mathbb{R} \) be weights for its edges \((i,j) \in E\). A Hamiltonian path between \( s, f \in V \) can be found by solving the following program, returning \( P = \{(i,j)|x_{ij} = 1\} \).

\[
\max_{x_{ij} \in \{0,1\} : (i,j) \in E} \sum_{i \neq j} c_{ij} x_{ij} \\
\text{s.t.} \quad \sum_{i=0,i \neq j} x_{ij} = 1 \quad \forall j \neq s; \quad \sum_{j=0,j \neq i} x_{ij} = 1 \quad \forall i \neq e; \\
\quad u_i - u_j + |V|x_{ij} \leq |V| - 1 \quad \forall (i,j) \in E
\]
References


