The stages of event extraction

David Ahn
Intelligent Systems Lab Amsterdam
University of Amsterdam
ahn@science.uva.nl

Abstract

Event detection and recognition is a complex task consisting of multiple sub-tasks of varying difficulty. In this paper, we present a simple, modular approach to event extraction that allows us to experiment with a variety of machine learning methods for these sub-tasks, as well as to evaluate the impact on performance these sub-tasks have on the overall task.

1 Introduction

Events are undeniably temporal entities, but they also possess a rich non-temporal structure that is important for intelligent information access systems (information retrieval, question answering, summarization, etc.). Without information about *what* happened, *where*, and to *whom*, temporal information about an event may not be very useful.

In the available annotated corpora geared toward information extraction, we see two models of events, emphasizing these different aspects. On the one hand, there is the TimeML model, in which an event is a word that points to a node in a network of temporal relations. On the other hand, there is the ACE model, in which an event is a complex structure, relating arguments that are themselves complex structures, but with only ancillary temporal information (in the form of temporal arguments, which are only noted when explicitly given). In the TimeML model, every event is annotated, because every event takes part in the temporal network. In the ACE model, only "interesting" events (events that fall into one of 34 predefined categories) are annotated.

The task of automatically extracting ACE events is more complex than extracting TimeML events (in line with the increased complexity of ACE events), involving detection of event anchors, assignment of an array of attributes, identification of arguments and assignment of roles, and determination of event coreference. In this paper, we present a modular system for ACE event detection and recognition. Our focus is on the difficulty and importance of each sub-task of the extraction task. To this end, we isolate and perform experiments on each stage, as well as evaluating the contribution of each stage to the overall task.

In the next section, we describe events in the ACE program in more detail. In section 3, we provide an overview of our approach and some information about our corpus. In sections 4 through 7, we describe our experiments for each of the subtasks of event extraction. In section 8, we compare the contribution of each stage to the overall task, and in section 9, we conclude.

2 Events in the ACE program

The ACE program[1] provides annotated data, evaluation tools, and periodic evaluation exercises for a variety of information extraction tasks. There are five basic kinds of extraction targets supported by ACE: entities, times, values, relations, and events. The ACE tasks for 2005 are more fully described in (ACE, 2005). In this paper, we focus on events, but since ACE events are complex structures involving entities, times, and values, we briefly describe these, as well.

ACE entities fall into seven types (person, organization, location, geo-political entity, facility, vehicle, weapon), each with a number of subtypes. Within the ACE program, a distinction is made between entities and entity mentions (similarly be-

[1] http://www.nist.gov/speech/tests/ace/

Proceedings of the Workshop on Annotating and Reasoning about Time and Events, pages 1–8,
Sydney, July 2006. ©2006 Association for Computational Linguistics

tween event and event mentions, and so on). An entity mention is a referring expression in text (a name, pronoun, or other noun phrase) that refers to something of an appropriate type. An entity, then, is either the actual referent, in the world, of an entity mention or the cluster of entity mentions in a text that refer to the same actual entity. The ACE Entity Detection and Recognition task requires both the identification of expressions in text that refer to entities (i.e., entity mentions) and coreference resolution to determine which entity mentions refer to the same entities.

There are also ACE tasks to detect and recognize times and a limited set of values (contact information, numeric values, job titles, crime types, and sentence types). Times are annotated according to the TIMEX2 standard, which requires normalization of temporal expressions (timexes) to an ISO-8601-like value.

ACE events, like ACE entities, are restricted to a range of types. Thus, not all events in a text are annotated—only those of an appropriate type. The eight event types (with subtypes in parentheses) are Life (Be-Born, Marry, Divorce, Injure, Die), Movement (Transport), Transaction (Transfer-Ownership, Transfer-Money), Business (Start-Org, Merge-Org, Declare-Bankruptcy, End-Org), Conflict (Attack, Demonstrate), Contact (Meet, Phone-Write), Personnel (Start-Position, End-Position, Nominate, Elect), Justice (Arrest-Jail, Release-Parole, Trial-Hearing, Charge-Indict, Sue, Convict, Sentence, Fine, Execute, Extradite, Acquit, Appeal, Pardon). Since there is nothing inherent in the task that requires the two levels of type and subtype, for the remainder of the paper, we will refer to the combination of event type and subtype (e.g., Life:Die) as the event type.

In addition to their type, events have four other attributes (possible values in parentheses): modality (Asserted, Other), polarity (Positive, Negative), genericity (Specific, Generic), tense (Past, Present, Future, Unspecified).

The most distinctive characteristic of events (unlike entities, times, and values, but like relations) is that they have arguments. Each event type has a set of possible argument roles, which may be filled by entities, values, or times. In all, there are 35 role types, although no single event can have all 35 roles. A complete description of which roles go with which event types can be found in the annotation guidelines for ACE events (LDC, 2005).

Events, like entities, are distinguished from their mentions in text. An event mention is a span of text (an *extent*, usually a sentence) with a distinguished *anchor* (the word that “most clearly expresses [an event’s] occurrence” (LDC, 2005)) and zero or more arguments, which are entity mentions, timexes, or values in the extent. An event is either an actual event, in the world, or a cluster of event mentions that refer to the same actual event. Note that the arguments of an event are the entities, times, and values corresponding to the entity mentions, timexes, and values that are arguments of the event mentions that make up the event.

The official evaluation metric of the ACE program is ACE value, a cost-based metric which associates a normalized, weighted cost to system errors and subtracts that cost from a maximum score of 100%. For events, the associated costs are largely determined by the costs of the arguments, so that errors in entity, timex, and value recognition are multiplied in event ACE value. Since it is useful to evaluate the performance of event detection and recognition independently of the recognition of entities, times, and values, the ACE program includes diagnostic tasks, in which partial ground truth information is provided. Of particular interest here is the diagnostic task for event detection and recognition, in which ground truth entities, values, and times are provided. For the remainder of this paper, we use this diagnostic methodology, and we extend it to sub-tasks within the task, evaluating components of our event recognition system using ground truth output of upstream components. Furthermore, in our evaluating our system components, we use the more transparent metrics of precision, recall, F-measure, and accuracy.

3 Our approach to event extraction

3.1 A pipeline for detecting and recognizing events

Extracting ACE events is a complex task. Our goal with the approach we describe in this paper is to establish baseline performance in this task using a relatively simple, modular system. We break down the task of extracting events into a series of classification sub-tasks, each of which is handled by a machine-learned classifier.

1. Anchor identification: finding event anchors (the basis for event mentions) in text and assigning them an event type;

2. Argument identification: determining which entity mentions, timexes, and values are arguments of each event mention;

3. Attribute assignment: determining the values of the modality, polarity, genericity, and tense attributes for each event mention;

4. Event coreference: determining which event mentions refer to the same event.

In principle, these four sub-tasks are highly interdependent, but for the approach described here, we do not model all these dependencies. Anchor identification is treated as an independent task. Argument finding and attribute assignment are each dependent only on the results of anchor identification, while event coreference depends on the results of all of the other three sub-tasks.

To learn classifiers for the first three tasks, we experiment with TiMBL[2], a memory-based (nearest neighbor) learner (Daelemans et al., 2004), and MegaM[3], a maximum entropy learner (Daumé III, 2004). For event coreference, we use only MegaM, since our approach requires probabilities. In addition to comparing the performance of these two learners on the various sub-tasks, we also experiment with the structure of the learning problems for the first two tasks.

In the remainder of this paper, we present experiments for each of these sub-tasks (sections 4– 7), focusing on each task in isolation, and then look at how the sub-tasks affect performance in the overall task (section 8). First, we discuss the preprocessing of the corpus required for our experiments.

3.2 Preprocessing the corpus

Because of restrictions imposed by the organizers on the 2005 ACE program data, we use only the ACE 2005 training corpus, which contains 599 documents, for our experiments. We split this corpus into training and test sets at the document-level, with 539 training documents and 60 test documents. From the training set, another 60 documents are reserved as a development set, which is used for parameter tuning by MegaM. For the remainder of the paper, we will refer to the 539 training documents as the training corpus and the 60 test documents as the test corpus.

For our machine learning experiments, we need a range of information in order to build feature vectors. Since we are interested only in performance on event extraction, we follow the methodology of the ACE diagnostic tasks and use the ground truth entity, timex2, and value annotations both for training and testing. Additionally, each document is tokenized and split into sentences using a simple algorithm adapted from (Grefenstette, 1994, p. 149). These sentences are parsed using the August 2005 release of the Charniak parser (Charniak, 2000)[4]. The parses are converted into dependency relations using a method similar to (Collins, 1999; Jijkoun and de Rijke, 2004). The syntactic annotations thus provide access both to constituency and dependency information. Note that with respect to these two sources of syntactic information, we use the word *head* ambiguously to refer both to the head of a constituent (i.e., the distinguished word within the constituent from which the constituent inherits its category features) and to the head of a dependency relation (i.e., the word on which the dependent in the relation depends).

Since parses and entity/timex/value annotations are produced independently, we need a strategy for matching (entity/timex/value) mentions to parses. Given a mention, we first try to find a single constituent whose offsets exactly match the extent of the mention. In the training and development data, there is an exact-match constituent for 89.2% of the entity mentions. If there is no such constituent, we look for a sequence of constituents that match the mention extent. If there is no such sequence, we back off to a single word, looking first for a word whose start offset matches the start of the mention, then for a word whose end offset matches the end of the mention, and finally for a word that contains the entire mention. If all these strategies fail, then no parse information is provided for the mention. Note that when a mention matches a sequence of constituents, the head of the constituent in the sequence that is shallowest in the parse tree is taken to be the (constituent) head of the entire sequence. Given a parse constituent, we take the entity type of that constituent to be the type of the smallest entity mention overlapping with it.

4 Identifying event anchors

4.1 Task structure

We model anchor identification as a word classification task. Although an event anchor may in principle be more than one word, more than 95% of

[2] `http://ilk.uvt.nl/timbl/`

[3] `http://www.isi.edu/~hdaume/megam/`

[4] `ftp://ftp.cs.brown.edu/pub/nlparser/`

the anchors in the training data consist of a single word. Furthermore, in the training data, anchors are restricted in part of speech (to nouns: NN, NNS, NNP; verbs: VB, VBZ, VBP, VBG, VBN, VBD, AUX, AUXG, MD; adjectives: JJ; adverbs: RB, WRB; pronouns: PRP, WP; determiners: DT, WDT, CD; and prepositions: IN). Thus, anchor identification for a document is reduced to the task of classifying each word in the document with an appropriate POS tag into one of 34 classes (the 33 event types plus a None class for words that are not an event anchor).

The class distribution for these 34 classes is heavily skewed. In the 202,135 instances in the training data, the None class has 197,261 instances, while the next largest class (Conflict:Attack) has only 1410 instances. Thus, in addition to modeling anchor identification as a single multi-class classification task, we also try to break down the problem into two stages: first, a binary classifier that determines whether or not a word is an anchor, and then, a multi-class classifier that determines the event type for the positive instances from the first task. For this staged task, we train the second classifier on the ground truth positive instances.

4.2 Features for event anchors

We use the following set of features for all configurations of our anchor identification experiments.

- Lexical features: full word, lowercase word, lemmatized word, POS tag, depth of word in parse tree
- WordNet features: for each WordNet POS category c (from N, V, ADJ, ADV):
  - If the word is in catgory c and there is a corresponding WordNet entry, the ID of the synset of first sense is a feature value
  - Otherwise, if the word has an entry in WordNet that is morphologically related to a synset of category c, the ID of the related synset is a feature value
- Left context (3 words): lowercase, POS tag
- Right context (3 words): lowercase, POS tag
- Dependency features: if the candidate word is the dependent in a dependency relation, the label of the relation is a feature value, as are the dependency head word, its POS tag, and its entity type
- Related entity features: for each entity/timex/value type t:
  - Number of dependents of candidate word of type t
  - Label(s) of dependency relation(s) to dependent(s) of type t
  - Constituent head word(s) of dependent(s) of type t
  - Number of entity mentions of type t reachable by some dependency path (i.e., in same sentence)
  - Length of path to closest entity mention of type t

4.3 Results

In table 1, we present the results of our anchor classification experiments (precision, recall and F-measure). The all-at-once conditions refer to experiments with a single multi-class classifier (using either MegaM or TiMBL), while the split conditions refer to experiments with two staged classifiers, where we experiment with using MegaM and TiMBL for both classifiers, as well as with using MegaM for the binary classification and TiMBL for the multi-class classification. In table 2, we present the results of the two first-stage binary classifiers, and in table 3, we present the results of the two second-stage multi-class classifiers on ground truth positive instances. Note that we always use the default parameter settings for MegaM, while for TiMBL, we set k (number of neighbors to consider) to 5, we use inverse distance weighting for the neighbors and weighted overlap, with information gain weighting, for all non-numeric features.

Both for the all-at-once condition and for multi-class classification of positive instances, the nearest neighbor classifier performs substantially better than the maximum entropy classifier. For binary classification, though, the two methods perform similarly, and staging either binary classifier with the nearest neighbor classifier for positive instances yields the best results. In practical terms, using the maximum entropy classifier for binary classification and then the TiMBL classifier to classify only the positive instances is the best solution, since classification with TiMBL tends to be slow.

	Precision	Recall	F
All-at-once/megam	0.691	0.239	0.355
All-at-once/timbl	0.666	0.540	0.596
Split/megam	0.589	0.417	0.489
Split/timbl	0.657	0.551	0.599
Split/megam+timbl	0.725	0.513	0.601

Table 1: Results for anchor detection and classification

	Precision	Recall	F
Binary/megam	0.756	0.535	0.626
Binary/timbl	0.685	0.574	0.625

Table 2: Results for anchor detection (i.e., binary classification of anchor instances)

5 Argument identification

5.1 Task structure

Identifying event arguments is a pair classification task. Each event mention is paired with each of the entity/timex/value mentions occurring in the same sentence to form a single classification instance. There are 36 classes in total: 35 role types and a None class. Again, the distribution of classes is skewed, though not as heavily as for the anchor task, with 20,556 None instances out of 29,450 training instances. One additional consideration is that no single event type allows arguments of all 36 possible roles; each event type has its own set of allowable roles. With this in mind, we experiment with treating argument identification as a single multi-class classification task and with training a separate multi-class classifier for each event type. Note that all classifiers are trained using ground truth event mentions.

5.2 Features for argument identification

We use the following set of features for all our argument classifiers.

- Anchor word of event mention: full, lowercase, POS tag, and depth in parse tree

	Accuracy
Multi/megam	0.649
Multi/timbl	0.824

Table 3: Accuracy for anchor classification (i.e., multi-class classification of positive anchor instances)

	Precision	Recall	F
All-at-once/megam	0.708	0.430	0.535
All-at-once/timbl	0.509	0.453	0.480
CPET/megam	0.689	0.490	0.573
CPET/timbl	0.504	0.535	0.519

Table 4: Results for arguments

- Event type of event mention
- Constituent head word of entity mention: full, lowercase, POS tag, and depth in parse tree
- Determiner of entity mention, if any
- Entity type and mention type (name, pronoun, other NP) of entity mention
- Dependency path between anchor word and constituent head word of entity mention, expressed as a sequence of labels, of words, and of POS tags

5.3 Results

In table 4, we present the results for argument identification. The all-at-once conditions refer to experiments with a single classifier for all instances. The CPET conditions refer to experiments with a separate classifier for each event type. Note that we use the same parameter settings for MegaM and TiMBL as for anchor classification, except that for TiMBL, we use the modified value difference metric for the three dependency path features.

Note that splitting the task into separate tasks for each event type yields a substantial improvement over using a single classifier. Unlike in the anchor classification task, maximum entropy classification handily outperforms nearest-neighbor classification. This may be related to the binarization of the dependency-path features for maximum entropy training: the word and POS tag sequences (but not the label sequences) are broken down into their component steps, so that there is a separate binary feature corresponding to the presence of a given word or POS tag in the dependency path.

Table 5 presents results of each of the classifiers restricted to Time-* arguments (Time-Within, Time-Holds, etc.). These arguments are of particular interest not only because they provide the link between events and times in this model of events, but also because Time-* roles, unlike other role

	Precision	Recall	F
All-at-once/megam	0.688	0.477	0.564
All-at-once/timbl	0.500	0.482	0.491
CPET/megam	0.725	0.451	0.556
CPET/timbl	0.357	0.404	0.379

Table 5: Results for Time-* arguments

	Accuracy
megam	0.795
timbl	0.793
baseline	0.802
majority (in training)	0.773

Table 6: Genericity

types, are available to all event types. We see that, in fact, the all-at-once classifiers perform better for these role types, which suggests that it may be worthwhile to factor out these role types and build a classifier specifically for temporal arguments.

6 Assigning attributes

6.1 Task structure

In addition to the event type and subtype attributes, (the event associated with) each event mention must also be assigned values for genericity, modality, polarity, and tense. We train a separate classifier for each attribute. Genericity, modality, and polarity are each binary classification tasks, while tense is a multi-class task. We use the same features as for the anchor identification task, with the exception of the lemmatized anchor word and the WordNet features.

6.2 Results

The results of our classification experiments are given in tables 6, 7, 8, and 9. Note that modality, polarity, and genericity are skewed tasks where it is difficult to improve on the baseline majority classification (Asserted, Positive, and Specific, respectively) and where maximum entropy and nearest neighbor classification perform very similarly. For tense, however, both learned classifiers perform substantially better than the majority baseline (Past), with the maximum entropy classifier providing the best performance.

	Accuracy
megam	0.750
timbl	0.759
baseline	0.738
majority (in training)	0.749

Table 7: Modality

	Accuracy
megam	0.955
timbl	0.955
baseline	0.950
majority (in training)	0.967

Table 8: Polarity

7 Event coreference

7.1 Task structure

For event coreference, we follow the approach to entity coreference detailed in (Florian et al., 2004). This approach uses a mention-pair coreference model with probabilistic decoding. Each event mention in a document is paired with every other event mention, and a classifier assigns to each pair of mentions the probability that the paired mentions corefer. These probabilities are used in a left-to-right entity linking algorithm in which each mention is compared with all already-established events (i.e., event mention clusters) to determine whether it should be added to an existing event or start a new one. Since the classifier needs to output probabilities for this approach, we do not use TiMBL, but only train a maximum entropy classifier with MegaM.

7.2 Features for coreference classification

We use the following set of features for our mention-pair classifier. The *candidate* is the earlier event mention in the text, and the *anaphor* is the later mention.

- *CandidateAnchor+AnaphorAnchor*, also POS tag and lowercase

	Accuracy
megam	0.633
timbl	0.613
baseline	0.535
majority (in training)	0.512

Table 9: Tense

	Precision	Recall	F
megam	0.761	0.580	0.658
baseline	0.167	1.0	0.286

Table 10: Coreference

- *CandidateEventType+AnaphorEventType*
- Depth of candidate anchor word in parse tree
- Depth of anaphor anchor word in parse tree
- Distance between candidate and anchor, measured in sentences
- Number, heads, and roles of shared arguments (same entity/timex/value w/same role)
- Number, heads, and roles of candidate arguments that are not anaphor arguments
- Number, heads, and roles of anaphor arguments that are not candidate arguments
- Heads and roles of arguments shared by candidate and anaphor in different roles
- *CandidateModalityVal+AnaphorModalityVal*, also for polarity, genericity, and tense

7.3 Results

In table 10, we present the performance of our event coreference pair classifier. Note that the distribution for this task is also skewed: only 3092 positive instances of 42,736 total training instances. Simple baseline of taking event mentions of identical type to be coreferent does quite poorly.

8 Evaluation with ACE value

Table 11 presents results of performing the full event detection and recognition task, swapping in ground truth (gold) or learned classifiers (learned) for the various sub-tasks (we also swap in majority classifiers for the attribute sub-task). For the anchor sub-task, we use the split/megam+timbl classifier; for the argument sub-task, we use the CPET/megam classifier; for the attribute sub-tasks, we use the megam classifiers; for the coreference sub-task, we use the approach outlined in section 7. Since in our approach, the argument and attribute sub-tasks are dependent on the anchor sub-task and the coreference sub-task is dependent on all of the other sub-tasks, we cannot freely swap in ground truth—e.g., if we use a learned classifier for the anchor sub-task, then there is no ground truth for the corresponding argument and attribute sub-tasks.

The learned coreference classifier provides a small boost to performance over doing no coreference at all (7.5% points for the condition in which all the other sub-tasks use ground truth (1 vs. 8), 0.6% points when all the other sub-tasks use learned classifiers (7 vs. 12)). From perfect coreference, using ground truth for the other subtasks, the loss in value is 11.4% points (recall that maximum ACE value is 100%). Note that the difference between perfect coreference and no coreference is only 18.9% points.

Looking at the attribute sub-tasks, the effects on ACE value are even smaller. Using the learned attribute classifiers (with ground truth anchors and arguments) results in 4.8% point loss in value from ground truth attributes (1 vs. 5) and only a 0.5% point gain in value from majority class attributes (4 vs. 5). With learned anchors and arguments, the learned attribute classifiers result in a 0.4% loss in value from even majority class attributes (3 vs. 7).

Arguments clearly have the greatest impact on ACE value (which is unsurprising, given that arguments are weighted heavily in event value). Using ground truth anchors and attributes, learned arguments result in a loss of value of 35.6% points from ground truth arguments (1 vs. 2). When the learned coreference classifier is used, the loss in value from ground truth arguments to learned arguments is even greater (42.5%, 8 vs. 10).

Anchor identification also has a large impact on ACE value. Without coreference but with learned arguments and attributes, the difference between using ground truth anchors and learned anchors is 22.2% points (6 vs. 7). With coreference, the difference is still 21.0% points (11 vs. 12).

Overall, using the best learned classifiers for the various subtasks, we achieve an ACE value score of 22.3%, which falls within the range of scores for the 2005 diagnostic event extraction task (19.7%–32.7%).[5] Note, though, that these scores are not really comparable, since they involve training on the full training set and testing on a separate set of documents (as noted above, the 2005 ACE testing data is not available for further experimentation, so we are using 90% of the original training data for training/development and

[5] For the diagnostic task, ground truth entities, values, and times, are provided, as they are in our experiments.

	anchors	args	attrs	coref	ACE value
1	gold	gold	gold	none	81.1%
2	gold	learned	gold	none	45.5%
3	learned	learned	maj	none	22.1%
4	gold	gold	maj	none	75.8%
5	gold	gold	learned	none	76.3%
6	gold	learned	learned	none	43.9%
7	learned	learned	learned	none	21.7%
8	gold	gold	gold	learned	88.6%
9	gold	gold	learned	learned	79.4%
10	gold	learned	gold	learned	46.1%
11	gold	learned	learned	learned	43.3%
12	learned	learned	learned	learned	22.3%

Table 11: ACE value

10% for the results presented here).

9 Conclusion and future work

In this paper, we have presented a system for ACE event extraction. Even with the simple breakdown of the task embodied by the system and the limited feature engineering for the machine learned classifiers, the performance is not too far from the level of the best systems at the 2005 ACE evaluation. Our approach is modular, and it has allowed us to present several sets of experiments exploring the effect of different machine learning algorithms on the sub-tasks and exploring the effect of the different sub-tasks on the overall performance (as measured by ACE value).

There is clearly a great deal of room for improvement. As we have seen, improving anchor and argument identification will have the greatest impact on overall performance, and the experiments we have done suggest directions for such improvement. For anchor identification, taking one more step toward binary classification and training a binary classifier for each event type (either for all candidate anchor instances or only for positive instances) may be helpful. For argument identification, we have already discussed the idea of modeling temporal arguments separately; perhaps introducing a separate classifier for each role type might also be helpful.

For all the sub-tasks, there is more feature engineering that could be done (a simple example: for coreference, boolean features corresponding to identical anchors and event types). Furthermore, the dependencies between sub-tasks could be better modeled.

References

2005. The ACE 2005 (ACE05) evaluation plan. `http://www.nist.gov/speech/tests/ace/ace05/doc/ace05-evalplan.v3.pdf`.

Eugene Charniak. 2000. A maximum-entropy-inspired parser. In *Proceedings of the 1st Meeting of NAACL*, pages 132–139.

Michael Collins. 1999. *Head-Driven Statistical Models for Natural Language Parsing*. Ph.D. thesis, University of Pennsylvania.

Walter Daelemans, Jakub Zavrel, Ko van der Sloot, and Antal van den Bosch. 2004. *TiMBL: Tilburg Memory Based Learner, version 5.1, Reference Guide*. University of Tilburg, ILK Technical Report ILK-0402. http://ilk.uvt.nl/.

Hal Daumé III. 2004. Notes on CG and LM-BFGS optimization of logistic regression. Paper available at `http://www.isi.edu/˜hdaume/docs/daume04cg-bfgs.ps`, August.

Radu Florian, Hany Hassan, Abraham Ittycheriah, Hongyan Jing, Nanda Kambhatla, Xiaoqiang Luo, Nicolas Nicolov, and Salim Roukos. 2004. A statistical model for multilingual entity detection and tracking. In *Proceedings of HLT/NAACL-04*.

Gregory Grefenstette. 1994. *Explorations in Automatic Thesaurus Discovery*. Kluwer.

Valentin Jijkoun and Maarten de Rijke. 2004. Enriching the output of a parser using memory-based learning. In *Proceedings of the 42nd Meeting of the ACL*.

Linguistic Data Consortium, 2005. *ACE (Automatic Content Extraction) English Annotation Guidelines for Events*, version 5.4.3 2005.07.01 edition.