To appear in the *Journal of Experimental & Theoretical Artificial Intelligence*
Vol. 00, No. 00, Month 20XX, 1–23

Adjudication of Coreference Annotations via Answer Set Optimization

Peter Schüller[a]*

[a]*Marmara University, Faculty of Engineering, Goztepe Kampusu, Istanbul, Turkey*

(*received March 14, 2017*)

We describe the first automatic approach for merging coreference annotations obtained from multiple annotators into a single gold standard. This merging is subject to certain linguistic hard constraints and optimization criteria that prefer solutions with minimal divergence from annotators. The representation involves an equivalence relation over a large number of elements. We use Answer Set Programming to describe two representations of the problem and four objective functions suitable for different datasets. We provide two structurally different real-world benchmark datasets based on the METU-Sabanci Turkish Treebank, and we report our experiences in using the Gringo, Clasp, and Wasp tools for computing optimal adjudication results on these datasets.

Keywords: Coreference Resolution, Adjudication, Answer Set Programming

1. Introduction

Coreference Resolution (Ng, 2010; Sapena, Padró, & Turmo, 2008) is the task of finding phrases in a text that refer to the same real-world entity. Coreference is commonly annotated by marking subsequences of tokens in the input text as *mentions* and putting sets of mentions into *chains* such that all mentions in a chain refer to the same, clearly identifiable entity in the world. For example in the text "*John is a musician. He played a new song. A girl was listening to the song. 'It is my favorite,' John said to her.*" (Lee et al., 2013) we can identify the following mentions.

$$\begin{array}{l} \text{[John]}^{(i)} \text{ is [a musician]}^{(ii)}\text{. [He]}^{(iii)} \text{ played [a new song]}^{(iv)}. \\ \text{[A girl]}^{(v)} \text{ was listening to [the song]}^{(vi)}. \\ \text{``[It]}^{(vii)} \text{ is } \big[\text{[my]}^{(ix)} \text{ favorite}\big]^{(viii)}\text{,'' [John]}^{(x)} \text{ said to [her]}^{(xi)}. \end{array} \qquad (1)$$

Roman superscripts denote mention IDs, chains in this text are as follows: $\{(i), (iii), (ix), (x)\}$ (John, He, my, John); $\{(iv), (vi), (vii)\}$ (a new song, the song, It); and $\{(v), (xi)\}$ (A girl, her), where roman numbers again refer to mention IDs. For building and testing automatic coreference resolution methods, annotated corpora, i.e., texts with mention and chain annotations, are an important resource. Once trained, coreference resolution systems can be applied in various applications, for example Cardie, Wiebe, Wilson, and Litman (2004) use it for complex opinion extraction, Mueller (2004) uses it for script-based story understanding, Witte, Khamis, and Rilling (2010) perform reasoning on coreference chains to create OWL Ontologies from texts, and Tuggener (2014) compares accuracy of

*Corresponding author. Email: peter.schuller@marmara.edu.tr
This work extends (Schüller, 2017) with a semi-automatic adjudication encoding, an extended discussion, a tool description, and several additional examples for objective functions and adjudication instances.

coreference resolution systems when used as preprocessing for discourse analysis, summarization, and finding entity contexts.

Adjudication is the task of combining mention and chain information from several human annotators into one single "gold standard" corpus. These annotations are often mutually conflicting and resolving these conflicts is a task that is *global* on the document level, i.e., it is not possible to decide the truth of the annotation of one token, mention, or chain, without considering other tokens, mentions, and chains in the same document.

We here present results and experiences obtained in a two-year project for creating a Turkish coreference corpus, which included an effort for developing and improving a (semi)automatic solution for coreference adjudication. We produced two datasets which are assembled from 475 individual annotations of 33 distinct documents from the METU-Sabanci Turkish Treebank (Say, Zeyrek, Oflazer, & Özge, 2004). Merging such annotations manually to create a gold standard is a tedious task. With sometimes more than 10 distinct annotations per document in our datasets, we required tool support. However, only manual adjudication tools such as BART (Versley et al., 2008) exist, because usually, only few annotations per document are collected and then manually adjudicated by an expert (for example coreference annotations in the OntoNotes corpus (Pradhan, Ramshaw, Weischedel, MacBride, & Micciulla, 2007) were created by at most two independent annotators per document).

To adjudicate a larger amount of annotations in an automatic way, we developed a (semi-)automatic solution based on Answer Set Programming (ASP) (Baral, 2004; Brewka, Eiter, & Truszczynski, 2011; Gebser, Kaminski, Kaufmann, & Schaub, 2012; Lifschitz, 2008). ASP is a logic programming and knowledge representation paradigm that allows for a declarative specification of problems and is suitable for solving large-scale combinatorial optimization problems.

Our contributions are as follows.

- We formalize the *problem* of coreference adjudication, introduce four *objective functions* that have practical relevance for both our datasets, and we describe the basic idea of semi-automatic adjudication in Section 3.
- We provide two *ASP encodings* in Section 4: the MM (mention-mention) encoding explicitly represents a transitive closure over the equivalence relation of chains, while the CM (chain-mention) encoding avoids this explicit representation. Moreover, we provide an ASP module for semi-automatic adjudication.
- We describe our tool and the intended adjudication workflow in Section 5.
- We describe and provide two real-life *datasets*,[1] outline their properties and differences, and report on *empirical experiments* unsatisfiable-core optimization and stratification using the tools Gringo (Gebser, Kaminski, König, & Schaub, 2011), Clasp (Gebser, Kaufmann, & Schaub, 2012), and Wasp (Alviano, Dodaro, Leone, & Ricca, 2015) in Section 6.
- We formulate *insights* about developing ASP applications, analyzing bottlenecks in such applications, and specific issues we encountered with ASP optimization in Section 7.

We describe related work in Section 8 and conclude in Section 9.

Our approach is not specific to Turkish, and we have integrated it in the publicly available CaspR tool[2] for performing (semi-)automatic adjudication of coreference annotations based on the popular CoNLL data format which can include coreference information.

[1]`https://bitbucket.org/knowlp/asp-coreference-benchmark`

[2]`https://bitbucket.org/knowlp/caspr-coreference-tool`

CaspR is the first automatic tool for coreference adjudication, and our datasets are the first published datasets for automatic adjudication, because usually only the final result of an adjudication process (i.e., the gold standard corpus) gets published.

2. Preliminaries

2.1. *Coreference Resolution*

Coreference resolution is the task of finding phrases in a text that refer to the same entity (Ng, 2010; Sapena et al., 2008). We call such phrases *mentions*, and we call a group of mentions that refers to one entity *chains*. Formally we can describe mention detection and coreference resolution as follows.

Given a document D which is a sequence of tokens $w_1, \ldots, w_n$, mention detection is the task of finding a set $M = \{(s_1, e_1), \ldots, (s_m, e_m)\}$ of mentions, where a mention (s, e) is a pair of indexes, $1 \leq s \leq e \leq n$, pointing to start token s and end token e in D.

Given a set M of mentions, coreference resolution is the task of partitioning M into a set of chains P such that all sequences of tokens $w_{s_i}, \ldots, w_{e_i}$ in all mentions (s_i, e_i) in one chain refer to the same entity.

Example 1 (Introduction example continued)**:** We have 31 tokens (including punctuation). Some of the tokens are $w_1 =$ 'John', $w_2 =$ 'is', ..., $w_{30} =$ 'her', $w_{31} =$ '.', the set of mentions is $M = \{(1,1), (3,4), (6,6), \ldots, (30,30)\}$, the correct chains are $P = \{\{(1,1), (6,6), (23,23), (27,27)\}, \ldots, \{(12,13), (30,30)\}\}$, where $(12,13)$ represents 'A girl' and $(30,30)$ represents 'her'. $\square$

Mentions can be part of other mentions, but in that case they (usually) cannot be coreferent. Moreover mentions are (usually) phrases: if m and m' are overlapping, m is properly contained in m' or vice versa.

Given a set of mentions, there are exponentially many potential solutions to the coreference resolution problem, and finding a globally optimal solution is NP-hard according to most measures of coreference optimality (Stoyanov & Eisner, 2012).

Note that *anaphora resolution* (Clark & González-Brenes, 2008; Hirst, 1981; Kehler, Kertz, Rohde, & Elman, 2008; Mitkov, 1999) is a different tasks than coreference resolution. The former deals only with references to earlier parts of a text and sometimes even only with references where a pronoun points to another phrase. Contrary to that, coreference resolution also deals with noun phrases that can refer to each other, and references can be in any direction within the text.

2.2. *Answer Set Programming*

ASP is a logic programming paradigm which is suitable for knowledge representation and finding solutions for computationally (NP-)hard problems (Brewka et al., 2011; Gelfond & Lifschitz, 1988; Lifschitz, 2008). We next give brief preliminaries of ASP programs with (uninterpreted) function symbols, aggregates, choices, and weak constraints. For a more elaborate description we refer to the ASP-Core-2 standard (Calimeri et al., 2012), or to books about ASP (Baral, 2004; Gebser, Kaminski, et al., 2012; Gelfond & Kahl, 2014).

Syntax. Let $\mathcal{C}$ and $\mathcal{X}$ be mutually disjoint sets of *constants* and *variables*, which we denote with first letter in lower case and upper case, respectively. Constant names are used for constant terms, predicate names, and names for uninterpreted functions. The set of *terms* $\mathcal{T}$ is recursively defined, it is the smallest set containing $\mathbb{N} \cup \mathcal{C} \cup \mathcal{X}$ as well as uninterpreted function terms of form $f(t_1,\dots,t_n)$ where $f \in \mathcal{C}$ and $t_1,\dots,t_n \in \mathcal{T}$. An *ordinary atom* is of the form $p(t_1,\dots,t_n)$, where $p \in \mathcal{C}$, $t_1,\dots,t_n \in \mathcal{T}$, and $n \geq 0$ is the *arity* of the atom. An *aggregate atom* is of the form $X = \#agg\{\, t : b_1,\dots,b_k \,\}$ with variable $X \in \mathcal{X}$, aggregation function $\#agg \in \{\#max, \#count\}$, with $1 < k$, $t \in \mathcal{T}$ and $b_1,\dots,b_k$ a sequence of atoms. A term or atom is *ground* if it contains no sub-terms that are variables.

A *rule* r is of the form $\alpha \leftarrow \beta_1,\dots,\beta_n, \textbf{not } \beta_{n+1},\dots,\textbf{not } \beta_m$ where $m \geq 0$, α is an ordinary atom, β_j, $0 \leq j \leq m$ is an atom, and we let $H(r) = \{\alpha\}$ and $B(r) = \{\beta_1,\dots,\beta_n, \textbf{not } \beta_{n+1},\dots,\textbf{not } \beta_m\}$. A *program* is a finite set P of rules. A rule r is a *constraint*, if $k = 0$ and $m \neq 0$, and a *fact* if $m = 0$.

A *weak constraint* is of form $\leftsquigarrow \beta_1,\dots,\beta_n, \textbf{not } \beta_{n+1},\dots,\textbf{not } \beta_m.\ [w@1, t_1,\dots,t_k]$ where all β_j are atoms, and all t_i are terms such that each variable in some t_i is contained in some β_j (note that 1 in $w@1$ shows the 'level' which we do not use).

Semantics.. Semantics of an ASP program P are defined using its Herbrand Base HB_P and its ground instantiation $grnd(P)$. An aggregate literal in the body of a rule accumulates truth values from a set of atoms, e.g., $N = \#count\{A : p(A)\}$ is true wrt. an interpretation $I \subseteq HB_P$ iff the extension of $p/1$ in I has size N. Using the usual notion of satisfying a rule with respect to a given interpretation, the FLP-reduct (Faber, Pfeifer, & Leone, 2011) fP^I reduces a program P using an answer set candidate I: $fP^I = \{r \in grnd(P) \mid I \models B(r)\}$. Finally I is an answer set of P ($I \in AS(P)$) iff I is a minimal model of fP^I. Weak constraints define, that an answer set I has cost equivalent to the term w for each distinct tuple $t_1,\dots,t_k$ of constraints that have a satisfied body wrt. I. Answer sets of the lowest cost are preferred.

Syntactic Sugar.. Anonymous variables of form '_' are replaced by new variable symbols. Choice constructions can occur instead of rule heads, they generate a set of candidate solutions if the rule body is satisfied; e.g., $1\{p(a); p(b)\} \leq 2$ in the rule head generates all solution candidates where at least 1 and at most 2 atoms of the set $\{p(a), p(b)\}$ are true (bounds can be omitted).

3. Automatic Coreference Adjudication

Coreference adjudication can be formalized as follows.

> Given a document D of tokens and $u \geq 2$ partitions $P_1,\dots,P_u$ of mentions in D, we search for a single set of mentions M and a partitioning P of M.

Clearly, annotations might be contradictory and we need to ensure certain structural constraints in the solution, moreover there can be multiple solutions and some will have better quality than others.

A solution might also contain equivalences between mentions that are not present in any annotation. This can happen, because chains are equivalence relations: if we merge equivalence relations that are not subsets of each other, the new equivalence relation is the reflexive, symmetric, and transitive closure of the original relations.

Example 2 (continued)**:** Assume that in parallel to chain $\{(i), (iii), (ix), (x)\}$ (John, He, my, John) we obtain from another annotator the chain $\{(i), (ii), (x)\}$ (John, a musician, John). If we merge these chains naively by merging the sets, we obtain a single chain $\{(i), (ii), (iii), (ix), (x)\}$ (John, a musician, He, my, John) although no annotator indicated that (ii) and (iii) ('a musician' and 'He') belong to the same chain.

Note, that mention pairs (i) and (ii), respectively (vii) and $(viii)$, are in a predicative relationship ('is'), therefore they are, per convention, not considered to be coreferent. □

```
1   (2
2 (3)
3  2)
4 (3)
5  -
6 (2)
```
(a) in1.conll

```
1  (3
2   -
3  3)
4   -
5 (3)
6   -
```
(b) in2.conll

```
1 (1(2)
2     -
3    1)
4   (2)
5     -
6   (1)
```
(c) in3.conll

```
1  (2
2 (4)
3  2)
4   -
5 (2)
6 (4)
```
(d) in4.conll

Figure 1. Minimalistic example CoNLL input files containing two columns: token index (first column) and coreference chains (second column).

```
1 (1(3)
2   (2)
3    1)
4   (3)
5   (1)
6   (2)
```
(a) [U]

```
1 (1(2)
2   (2)
3    1)
4   (2)
5   (1)
6   (1)
```
(b) [UA]

```
1  (1     1  (1
2   -     2   -
3  1)     3  1)
4   -     4   -
5   -     5 (1)
6 (1)     6   -
```
(c) [V]

```
1  (1
2   -
3  1)
4   -
5 (1)
6 (1)
```
(d) [VA]

Figure 2. Optimal solutions of automatic adjudication of inputs in Figure 1 with all four objectives.

Next we give a symbolic and more comprehensive running example.

Example 3: Figure 1 gives an adjudication problem consisting of input from four annotators. We show token index in the first column and coreference chains in the second column of the CoNLL format, where single-token mentions are marked as '(X)', and multi-token mentions start with '(X' and end with 'X)'. In these coreference markers, X is the chain the mention belongs to.

In this example, annotator (a) indicated a chain with ID 2 containing mentions $\{(1,3),(6,6)\}$ as well as a chain with ID 3 containing mentions $\{(2,2),(4,4)\}$.

There are several possible outcomes for merging these (input) annotations into a single (output) gold standard annotation. Intuitively, annotators (a) and (c) indicated that mention $(1,3)$ is coreferent with $(6,6)$, while annotators (b) and (d) indicated that $(1,3)$ is coreferent with $(5,5)$. Only annotator (c) marked $(1,1)$ as a separate mention that is coreferent with $(4,4)$. □

In the following we describe objective functions for selecing preferred solutions.

3.1. *Objective Functions*

We next represent input and output chains as sets of links between pairs of mentions (mention-mention links). We define a preference relation for filtering out undesired solutions where we incur cost under the following conditions.

(C1) Not using a link provided by an annotator in the solution.
(C2) Using a link provided by an annotator where a number of other annotators did not provide the same link.
(C3) Putting two mentions into a chain where no annotator gave any evidence for this (see Example 2).

We incur separate cost for each annotator who provided a link in (C1), and for each annotator who did not provide a link that was used in (C2).

Concretely we use the following objective functions in our application:

`[V]` Cost 2 for (C1), cost 1 for (C2), and (C3) is a hard constraint (infinite cost).
`[U]` Cost 2 for (C1), no cost for (C2), and (C3) is a hard constraint.
`[VA]` Cost 2 for (C1), cost 1 for (C2), and cost 1 for (C3).
`[UA]` Cost 2 for (C1), no cost for (C2), and cost 1 for (C3).

Note that cost for (C2) is higher than for (C1) because we observed that annotators miss links more frequently than they add spurious links. Moreover, using cost 1 for (C1) in `[U]` would yield the same preference, however we use cost 2 to obtain a more uniform ASP encoding.

Intuitively, objectives containing letter `V` apply "voting" to prefer certain solutions: a link given by only one annotator can be dismissed if many other annotators do not give the same link. On the other hand, objectives with letter `U` use "as many mentions as possible": there is no cost for (C2). Finally, objectives containing letter `A` allow additioal links at a cost.

Example 4 (continued from Example 3)**:** Figure 2 shows automatically adjudicated results for all objective functions, based on annotations given in Figure 1.

Objective `[U]` in 2(a) produces a single optimal adjudication solution where both chains from 1(a) and chain 1 from 1(c) is ignored, which yields a total cost of 6 and ignores three mention-mention links.

Objective `[UA]` in 2(b) produces a single optimal adjudication solution of cost 4: chain 4 from 1(d) is ignored (cost 2) and two links that are not present in any annotation are created (each incurs cost 1): between $(1,1)$ and $(2,2)$, and between $(5,5)$ and $(6,6)$.

Objective `[V]` produces two optimal adjudication solutions, each with cost 12. The left solution in 2(c) uses chain 2 from 1(a), which is equivalent to chain 1 in 1(c). This incurs a cost of 2 because links of this chain are absent in 1(b) and 1(d). All other chains (one each in 1(a), 1(b), and 1(c), two in 1(d)) are ignored at a total cost of 10 (each of these chains contains one link). Using any of these ignored chains would incur a cost of 3 per chain because the chain was not given by three out of four annotators. Using chain 3 in 1(b) or chain 2 in 1(d) would create an additional link between $(5,5)$ and $(6,6)$, which is not permitted in objective `[V]`. The second solution (2(c) right) is symmetric to the first one: it uses only chain 3 from 1(b), which is equivalent to chain 2 in 1(d).

Objective `[VA]` produces a single optimal solution 2(d) of cost 11. Intuitively, this solution merges both solutions of `[V]`. Using chain 2 from 1(a), chain 3 from 1(b), chain 1 from 1(c), and chain 2 from 1(d) incurs cost 4 because each link is not annotated by two out of four annotators. Moreover, three links (chains 3 in 1(a), chain 2 in 1(c), and chain 4 in 1(d)) are ignored, which contributes an additional cost of 6. The link between $(5,5)$ and $(6,6)$, which was not specified by any annotator, incurs a cost of 1. □

The main idea of these preferences is to use the given information optimally while producing an overall consistent solution. The preferences are motivated by properties of our datasets: if mentions are given to annotators, they only disagree on assignment of mentions to chains, and `[V]` and `[VA]` make sure that the result reflects the opinion of the majority of annotators. Contrarily, if mentions are not given, annotators must produce mentions and chains, and often disagree on mentions, such that `[V]` and `[VA]` eliminate most mentions and chains completely. In the latter case, `[U]`and `[UA]` combined with a semi-automatic workflow is the better choice.

```
1  (2   (3 (1(2)  (2   =(1
2 (3)   -      - (4)   (2)
3  2)  3)     1)  2)   =1)
4 (3)   -    (2)   -   (3)
5   - (3)      - (2)   (1)
6 (2)   -    (1) (4)  =(1)
```

(a) Full output of automatic adjudication after manually enforcing mentions $(1, 3)$ and $(6, 6)$ in chain 1.

```
1  (2   (3 (1(2)  (2   =(1
2 (3)   -      - (4)   (2)
3  2)  3)     1)  2)   =1)
4 (3)   -    (2)   -   (2)
5   - (3)      - (2)     -
6 (2)   -    (1) (4)  =(1)
```

(b) Result of performing automatic adjudication relative to enforced coreference information in (a).

Figure 3. Semi-automatic adjudication of annotations given in Figure 1 using objective `[U]`. Manually enforced coreference information in the last column of the CoNLL format is prefixed with '`=`'.

3.2. *Semi-automatic Adjudication*

Automatic adjudication works well if enough annotations with high inter-annotator-agreement are available. Otherwise, automatic adjudication is merely a preprocessing step for human adjudication. In case of low amounts of annotations per document, or highly divergent annotator opinions, making the final decision on correctness of an annotation should be done by a human expert.

For that purpose, we allow a partial specification of chain and mention information as additonal input. We extend our approach so that it produces an optimal solution relative to this given information.

In practice, we do this by producing a human readable output format in automatic adjudication. This format is in the popular CoNLL format (see Figure 3) and contains tokens and coreference information from all annotators and from automatic adjudication. The idea is that a human adjudicator can inspect the output and manually specify parts of the output, followed by re-optimization of the adjudication relative to these manually enforced parts of the output (more details about this is given in Section 5).

Example 5: Figure 3 shows a CoNLL file that was generated from automatic adjudication of annotations in Figure 1 using objective `[U]`. The rightmost column originally was the same as in Figure 2(a) but the human adjudicator has specified the following coreference information (prefixed with '`=`'): mentions $(1, 3)$ and $(6, 6)$ must be in a chain, and mention $(1, 1)$, which was coreferent with $(4, 4)$, does not exist.

Re-optimizing Figure 3(a) yields Figure 3(b) where the result of automatically adjudicating relative to enforced mention and chain information is put into the last column. Different from the adjudication result explained in Example 4, coreference between $(1, 3)$ and $(5, 5)$ is no longer possible because it would generate a non-annotated link between $(5, 5)$ and $(6, 6)$. Therefore token 5 carries no mention information ('`-`'). Moreover, the link between $(1, 1)$ and $(4, 4)$ is ruled out because mention $(1, 1)$ was enforced to be absent, which makes the link between $(2, 2)$ and $(4, 4)$ appear in the optimal solution. $\square$

4. ASP Encodings

We next describe input and output representations for our adjudication program. Then we provide two ASP encodings, which model coreference adjudication and assign cost to solutions according to Section 3.1. The MM Encoding explicitly represents the transitive closure of mention-mention links, while the CM encoding avoids this representation by assuming an upper limit on the number of chains and guessing which mention belongs to which chain.

In all encodings, we use the convention that variables C, A, Am, M, S, and E, will be substituted with chain IDs, annotator IDs, annotator mention IDs, canonical mentions of form $mid(S, E)$, start token indexes, and end token indexes, respectively. We also use subscripted versions of these variables.

4.1. *Input and Output ASP Representation*

Given annotator inputs $P_1, \ldots, P_u$ where each chain $C_c \in P_a$ is a set of mentions of form (s_j, e_j), we represent that annotator put mention (s_j, e_j) into chain C_c as a fact $cm(a, c, j)$, moreover we represent the mention itself as a fact $mention(a, j, s_j, e_j)$. For each annotator a, constants c and m uniquely represent chains and mentions of that annotator, respectively.

Example 6 (continued)**:** Tokens in Example 1 are numbered with integers (token indexes) as follows.

$$\text{John}^1 \text{ is}^2 \text{ a}^3 \text{ musician}^4 \text{ .}^5 \text{ He}^6 \text{ played}^7 \text{ a}^8 \text{ new}^9 \text{ song}^{10} \text{ .}^{11}$$

Consider that annotator a_1 created chain $c_1 = \{a, b, c\}$ containing mentions $a = (1, 1)$ for 'John', $b = (3, 4)$ for 'a musician', and $c = (6, 6)$ for 'He'. Annotator a_2 annotated chain $c_2 = \{d, e\}$ containing mentions $d = (4, 4)$ for 'musician' and $e = (6, 6)$ for 'He'. These annotations are represented in ASP as follows:

$$\begin{array}{lll} mention(a_1, a, 1, 1). & mention(a_1, b, 3, 4). & mention(a_1, c, 6, 6). \\ cm(a_1, c_1, a). & cm(a_1, c_1, b). & cm(a_1, c_1, c). \\ mention(a_2, d, 4, 4). & mention(a_2, e, 6, 6). & \\ cm(a_2, c_2, d). & cm(a_2, c_2, e). & \end{array}$$

where constants c_1 and c_2 are IDs for representing chains c_1 and c_2, respectively, and similarly constants $a, \ldots, e$ are IDs for representing mentions $a, \ldots, e$. □

The answer sets of our logic program represent a set of chains without annotator information, represented as atoms of the form $result_{cm}(C, mid(S, E))$, which indicates that in chain C there is a mention that is a span from start token S to end token E, inclusive.

Example 7 (continued)**:** Assume we have merged the annotations of the previous example into two chains $v = \{(1, 1), (3, 4)\}$ and $w = \{(4, 4), (6, 6)\}$, then this would be represented by the following atoms:

$$\begin{array}{ll} result_{cm}(v, mid(1, 1)) & result_{cm}(v, mid(3, 4)) \\ result_{cm}(w, mid(4, 4)) & result_{cm}(w, mid(6, 6)) \end{array}$$

Intuitively, this would mean that 'John' and 'a musician' are the same entity, as well as 'musician' and 'He', but 'John' is not the same as entity as 'He'. □

Input for *semi-automatic adjudication* consists of mentions, chains, and moreover the absence of any mention annotation can be specified for individual tokens. Mentions and chains are represented as a separate annotation with a special annotator ID '*forced*'. Empty tokens are represented as facts $empty(Token)$.

$$\begin{aligned}
\mathit{na}(N) &\leftarrow N = \#\mathit{count}\,\{\, A : \mathit{mention}(A, \_, \_, \_)\,\}. && (2)\\
\mathit{acmen}(A, \mathit{Am}, \mathit{mid}(S, E)) &\leftarrow \mathit{mention}(A, \mathit{Am}, S, E). && (3)\\
\mathit{asamechain}(A, \mathit{Am}_1, \mathit{Am}_2) &\leftarrow \mathit{cm}(A, C, \mathit{Am}_1),\ \mathit{cm}(A, C, \mathit{Am}_2),\ \mathit{Am}_1 \neq \mathit{Am}_2. && (4)\\
\mathit{csamechain}(M_1, M_2) &\leftarrow \mathit{acmen}(A, \mathit{Am}_1, M_1),\ \mathit{acmen}(A, \mathit{Am}_2, M_2),\\
&\qquad \mathit{asamechain}(A, \mathit{Am}_1, \mathit{Am}_2),\ M_1 < M_2. && (5)\\
\mathit{evidence}(M_1, M_2, \mathit{Ev}) &\leftarrow \mathit{csamechain}(M_1, M_2),\ \mathit{Ev} = \#\mathit{count}\,\{\\
&\qquad F : \mathit{asamechain}(A, \mathit{Am}_1, \mathit{Am}_2),\ \mathit{Am}_1 < \mathit{Am}_2,\\
&\qquad\quad \mathit{acmen}(A, \mathit{Am}_1, M_1),\ \mathit{acmen}(A, \mathit{Am}_2, M_2)\\
&\qquad \},\ \mathit{Ev} > 0. && (6)\\
\mathit{cmomitcost}(M_1, M_2, 2 \cdot K) &\leftarrow \mathit{csamechain}(M_1, M_2),\ \mathit{evidence}(M_1, M_2, K). && (7)\\
\mathit{cmusecost}(M_1, M_2, N - K) &\leftarrow \mathit{csamechain}(M_1, M_2),\ \mathit{evidence}(M_1, M_2, K),\\
&\qquad N - K > 0,\ \mathit{na}(N). && (8)
\end{aligned}$$

Figure 4. Common deterministic definitions used in objective function constraints.

Example 8 (continued)**:** In Figure 3(a) the human adjudicator specified mentions $(1, 3)$ to be coreferent with $(6, 6)$. This is represented by the following facts.

$$\mathit{mention}(\mathit{forced}, m_1, 1, 3) \leftarrow . \qquad \mathit{mention}(\mathit{forced}, m_2, 6, 6) \leftarrow .$$
$$\mathit{cm}(\mathit{forced}, c_1, m_1) \leftarrow . \qquad \mathit{cm}(\mathit{forced}, c_1, m_2) \leftarrow .$$

where the chain is represented by constant c_1 and the mentions by m_1 and m_2. □

We next describe our encodings.

4.2. *Common Rules for Objective Costs*

Figure 4 shows a common program module which defines helpful concepts for realizing cost aspects (C1)–(C3) from Section 3.1. The rules depend only on input facts, hence this module is deterministic.

Rule (2) represents the number of annotations in the input in *na*/1. This number is used for relating the weight of one annotator's input compared with all annotators. Rule (3) defines predicate *acmen*/3 which connects annotated mentions with their canonical representation in a term of form $\mathit{mid}(S, E)$ where S and E are the starting and ending token of the mention in the document. Rule (4) represents in *asamechain*/3 which mentions of each annotator are in the same chain, and rule (5) represents the same in *csamechain*/2 for canonical mentions across all annotators.

Based on *csamechain* and *asamechain*, rule (6) represents for each pair of canonical mentions (M_1, M_2) that are potentially in the same chain, how many annotators actually provided evidence for putting them into the same chain. This value is represented only if it is above zero. From this evidence, rule (7) defines cost component (C1) which is a cost of 2 for each annotator who provided positive evidence for the respective link. Similarly, using the total number of annotators, rule (8) defines cost component (C2) which is a cost of 1 for each annotator who did not put both canonical mentions M_1 and M_2 into the same chain, while at least one other annotator did so.

$$
\begin{aligned}
\mathit{link}(A, Am_1, Am_2) &\leftarrow \mathit{cm}(A, C, Am_1),\ \mathit{cm}(A, C, Am_2),\ Am_1 < Am_2. && (9)\\
\{\ \mathit{uselink}(A, Am_1, Am_2)\ \} &\leftarrow \mathit{link}(A, Am_1, Am_2). && (10)\\
\mathit{clink}(M_1, M_2) &\leftarrow \mathit{uselink}(A, Am_1, Am_2),\ M_1 < M_2,\\
&\qquad \mathit{acmen}(A, Am_1, M_1),\ \mathit{mention}(A, Am, M_2). && (11)\\
\mathit{clink}(M_1, M_2) &\leftarrow \mathit{uselink}(A, Am_2, Am_1),\ M_1 < M_2,\\
&\qquad \mathit{acmen}(A, Am_1, M_1),\ \mathit{mention}(A, Am_2, M_2). && (12)\\
\mathit{cc}(M_1, M_2) &\leftarrow \mathit{clink}(M_1, M_2). && (13)\\
\mathit{cc}(M_2, M_1) &\leftarrow \mathit{cc}(M_1, M_2). && (14)\\
\mathit{cc}(M_1, M_3) &\leftarrow \mathit{cc}(M_1, M_2),\ \mathit{cc}(M_2, M_3). && (15)\\
\mathit{notrepresentative}(M_2) &\leftarrow \mathit{cc}(M_1, M_2),\ M_1 < M_2. && (16)\\
\mathit{result}_{\mathit{cm}}(M_1, M_2) &\leftarrow \mathit{cc}(M_1, M_2),\ \textbf{not}\ \mathit{notrepresentative}(M_1). && (17)\\
\mathit{resultchain}(C) &\leftarrow \mathit{result}_{\mathit{cm}}(C, \_). && (18)\\
&\leftarrow \mathit{resultchain}(C),\ \#\mathit{count}\ \{\ M : \mathit{result}_{\mathit{cm}}(C, M)\ \} \leq 1. && (19)\\
&\leftarrow \mathit{result}_{\mathit{cm}}(C, \mathit{mid}(S_1, E_1)),\ \mathit{result}_{\mathit{cm}}(C, \mathit{mid}(S_2, E_2)),\\
&\qquad S_1 \leq S_2,\ E_2 \leq E_1,\ (S_1, E_1) \neq (S_2, E_2). && (20)
\end{aligned}
$$

Figure 5. MM encoding.

$$
\begin{aligned}
&\leftsquigarrow \textbf{not}\ \mathit{clink}(M_1, M_2), \mathit{cmomitcost}(M_1, M_2, \mathit{Cost}). && [\mathit{Cost}@1, M_1, M_2, \mathit{omit}] && (21)\\
&\leftsquigarrow \mathit{clink}(M_1, M_2), \mathit{cmusecost}(M_1, M_2, \mathit{Cost}). && [\mathit{Cost}@1, M_1, M_2, \mathit{use}] && (22)\\
&\leftarrow \mathit{cc}(M_1, M_2), M_1 < M_2, \textbf{not}\ \mathit{clink}(M_1, M_2). && && (23)\\
&\leftsquigarrow \mathit{cc}(M_1, M_2), M_1 < M_2, \textbf{not}\ \mathit{clink}(M_1, M_2). && [1@1, M_1, M_2] && (24)
\end{aligned}
$$

Figure 6. MM objective function variations.

4.3. *Mention-Mention Encoding (MM)*

Figure 5 shows the core module of the MM encoding. Rule (9) represents annotated mention-mention links in predicate *link*/3, and rule (10) is a guess whether to use each of these links. Used links are represented as canonical mentions (*cmentions*) in rules (11) and (12) such that their annotator information is projected away.[3] Note that these rules make use of deterministically defined predicate *acmen*/3 from the common program module given in Figure 4. Rules (13)–(15) represent the reflexive, symmetric, and transitive closure of *clink/2* in *cc/2*. This represents result chains in an explicit equivalence relation over cmentions: each strongly connected component (SCC) of *cc/2* is equivalent to a coreference chain. Rule (16) defines which elements of the equivalence relation is not the lexicographically smallest element in the SCC, and rule (17) defines result chains by using the smallest cmention in each SCC as representative for the corresponding chain. Rule (18) defines that these cmentions represent chains. Constraint (19) enforces that each chain contains at least two mentions. Moreover constraint (20) requires that no chain contains two mentions where one is contained in the other (this is linguistically

[3]This requires two rules, because mention and cmention IDs can have different lexicographic order.

Table 1. Objective functions and (weak) constraints that are used to realize them in MM and CM encodings.

	Encoding MM				Encoding CM			
(Weak) constraint	`[V]`	`[U]`	`[VA]`	`[UA]`	`[V]`	`[U]`	`[VA]`	`[UA]`
(21) = (35)	X	X	X	X	X	X	X	X
(22) = (36)	X		X		X		X	
(23)	X	X						
(24)			X	X				
(37)					X	X		
(38)							X	X

$$countchain(A,N) \leftarrow cm(A,\_,\_),\ N = \#count\,\{\,C : cm(A,C,\_)\,\}. \tag{25}$$
$$maxchain(N) \leftarrow N = \#max\,\{\,C : countchain(\_,C)\,\}. \tag{26}$$
$$chainlimit(6/5 \cdot N) \leftarrow maxchain(N). \tag{27}$$
$$\{\,resultchain(C) : C \in \{1,\dots,Max\}\,\} \leftarrow chainlimit(Max). \tag{28}$$
$$resultchain(C_2) \leftarrow resultchain(C_1),\ C_2 \in \{1,\dots,C_2-1\}. \tag{29}$$
$$cmention(M) \leftarrow clink(M,\_). \tag{30}$$
$$cmention(M) \leftarrow clink(\_,M). \tag{31}$$
$$1\,\{\,result_{cm}(C,M) : resultchain(C)\,\}\,1 \leftarrow cmention(M). \tag{32}$$
$$\leftarrow clink(M_1,M_2),\ result_{cm}(C,M_1),\ \mathbf{not}\ result_{cm}(C,M_2). \tag{33}$$
$$\leftarrow clink(M_1,M_2),\ \mathbf{not}\ result_{cm}(C,M_1),\ result_{cm}(C,M_2). \tag{34}$$

Figure 7. CM encoding. Additionally, rules (9)–(12) and constraints (19)–(20) from MM encoding are used.

motivated).

Figure 6 shows constraint variations for realizing objective functions. Weak constraint (21) incurs cost for non-existing links in *clink*(M_1, M_2), corresponding to cost (C1) in Section 3.1. Similarly (22) incurs cost for existing links, corresponding to cost (C2). Finally, corresponding to (C3), constraint (23) strictly forbids to put two mentions into a chain if there is no evidence for that from annotators, and weak constraint (24) alternatively incurs a cost for such mention pairs.

Table 1 shows the combinations of constraints that realize each objective function. Intuitively, we always incur cost for ignoring presence of mention-mention links via (21), sometimes we additionally incur cost for absence via (22). For those mention-mention links in the solution that were not annotated at all, we either forbid them completely via (23) or we incur an additional cost via (24).

4.4. *Chain-Mention Encoding (CM)*

Figure 7 shows the core of the CM encoding, which reuses some rules from MM encoding and directly guesses *resultcm/2* instead of deriving it from *clink/2*.

First we have a deterministic representation part. Rule (25) counts the number of chains per annotator, (26) finds the maximum number of chains that any of the annotators created, and (27) multiplies this number with 6/5. This value serves as an assumption for the maximum amount of chains in the result and deterministically depends on the given

$$\leftsquigarrow \textbf{not}\ \mathit{clink}(M_1, M_2), \mathit{cmomitcost}(M_1, M_2, \mathit{Cost}). \qquad [\mathit{Cost}@1, M_1, M_2, \mathit{omit}] \tag{35}$$

$$\leftsquigarrow \mathit{clink}(M_1, M_2), \mathit{cmusecost}(M_1, M_2, \mathit{Cost}). \qquad [\mathit{Cost}@1, M_1, M_2, \mathit{use}] \tag{36}$$

$$\leftarrow \mathit{result}_{cm}(C, M_1),\ \mathit{result}_{cm}(C, M_2),\ M_1 < M_2,\ \textbf{not}\ \mathit{clink}(M_1, M_2). \tag{37}$$

$$\leftsquigarrow \mathit{result}_{cm}(C, M_1),\ \mathit{result}_{cm}(C, M_2),\ M_1 < M_2,\ \textbf{not}\ \mathit{clink}(M_1, M_2). \qquad [1@1, M_1, M_2] \tag{38}$$

Figure 8. CM objective function variations.

instance.

Rule (28) uses this assumption on the number of chains to nondeterministically guess which chains exist in the result. Rule (29) performs symmetry breaking on this guess to use a gap-free sequence of chain IDs starting at 1.

We reuse rules (9)–(12) from the MM encoding for guessing which link to use from which annotator, and for canonicalizing these links in *clink/2*. Rules (30) and (31) represent cmentions that appear in the canonical links (in *clink*/2). In rule (32) we guess which cmention is contained in which result chain. Constraints (33) and (34) ensure, that links in *clink/2* which were selected in (10), are fully represented by *resultcm/2*.

Finally, we reuse rules (19)–(20) from the MM encoding for enforcing structural constraints.

We found that an assumption of a maximum number of chains can safely be used in practice, because annotators are consistent in the amount of annotations they produce over the whole document, i.e., annotators either create many chains or few chains relative to other annotators, they never create many chains in some part of the document and few chains in other parts of the same document relative to other annotators.

Figure 8 shows objective functions for CM. Weak constraints (35) and (36) for cost components (C1) and (C2) are the same as (21) and (22) in the MM encoding. Constraints for realizing (C3) are different, because in the CM encoding we have no transitive closure *cc*/2 at our disposal. Therefore, in constraint (37) we need to make an explicit join over *resultcm*/2 to rule out mention-mention links in the result that have no corresponding link in *clink*/2. Weak constraints (38) alternatively incur a cost for such links.

4.5. *Semi-automatic Encoding Module*

So far we have covered only automatic adjudication. Semi-automatic adjudication is based on enforced mention and chain information, and on tokens that are enforced to be empty as described in Section 4.1.

Figure 9 contains an encoding module for semi-automatic adjudication which can be used in addition to the CM or MM encodings and in combination with any of the objective functions. To ensure that annotations with special annotator ID '*forced*' are reproduced in every solution, we first represent several auxiliary concepts. Rules (39)–(41) represent which tokens contain (en-)forced annotations, rule (42) represents pairs of canonical mentions that must be part of the same chain, and rule (43) represents pairs that must be part of different chains. Constraint (44) ensures, that no canonical mention starts at an enforced token if that mention was not enforced, and (45) ensures the same for ends of

$$
\begin{aligned}
\mathit{force}_{\mathit{tok}}(\mathit{Token}) &\leftarrow \mathit{empty}(\mathit{Token}). &(39)\\
\mathit{force}_{\mathit{tok}}(S) &\leftarrow \mathit{mention}(\mathit{forced},\_,S,E). &(40)\\
\mathit{force}_{\mathit{tok}}(E) &\leftarrow \mathit{mention}(\mathit{forced},\_,S,E). &(41)\\
\mathit{force}_{\mathit{same}}(M_1,M_2) &\leftarrow \mathit{cm}(\mathit{forced},C,\mathit{Am}_1),\ \mathit{cm}(\mathit{forced},C,\mathit{Am}_2),\ \mathit{Am}_1 \neq \mathit{Am}_2,\ M_1 < M_2,\\
&\qquad \mathit{acmen}(\mathit{forced},\mathit{Am}_1,M_1),\ \mathit{acmen}(\mathit{forced},\mathit{Am}_2,M_2). &(42)\\
\mathit{force}_{\mathit{diff}}(M_1,M_2) &\leftarrow \mathit{cm}(\mathit{forced},C_1,\mathit{Am}_1),\ \mathit{cm}(\mathit{forced},C_2,\mathit{Am}_2),\ C_1 < C_2,\\
&\qquad \mathit{acmen}(\mathit{forced},\mathit{Am}_1,M_1),\ \mathit{acmen}(\mathit{forced},\mathit{Am}_2,M_2). &(43)\\
&\leftarrow \mathit{force}_{\mathit{tok}}(S),\ \mathit{result}_{\mathit{cm}}(\_,\mathit{mid}(S,E)),\ \textbf{not}\ \mathit{mention}(\mathit{forced},\_,S,E). &(44)\\
&\leftarrow \mathit{force}_{\mathit{tok}}(E),\ \mathit{result}_{\mathit{cm}}(\_,\mathit{mid}(S,E)),\ \textbf{not}\ \mathit{mention}(\mathit{forced},\_,S,E). &(45)\\
&\leftarrow \mathit{mention}(\mathit{forced},\_,S,E),\ \textbf{not}\ \mathit{result}_{\mathit{cm}}(\_,\mathit{mid}(S,E)). &(46)\\
&\leftarrow \mathit{force}_{\mathit{same}}(M_1,M_2),\ \mathit{result}_{\mathit{cm}}(C_1,M_1),\ \mathit{result}_{\mathit{cm}}(C_2,M_2),\ C_1 \neq C_2. &(47)\\
&\leftarrow \mathit{force}_{\mathit{diff}}(M_1,M_2),\ \mathit{result}_{\mathit{cm}}(C,M_1),\ \mathit{result}_{\mathit{cm}}(C,M_2). &(48)
\end{aligned}
$$

Figure 9. Program module for semi-automatic mode that allows to enforce certain mentions and chains, moreover empty tokens, i.e., tokens without coreference information.

mentions. Note that these two constraints and (39) ensure, that enforced empty tokens obtain no mention annotations.

Constraint (46) ensures, that all enforced mentions exist in the solution as canonical mentions. Finally, constraint (47) ensures, that mentions that should be in the same chain are not in different chains, and constraint (48) ensures, that mentions that should be in distinct chains are not in the same chain.

Adding these rules to encoding MM or CM is sufficient for ensuring that only solutions that reproduce the user-specified annotations remain as answer sets.

5. Tool and Adjudication Workflow

We have implemented our method in an open source tool called CaspR which is available publicly at `https://bitbucket.org/knowlp/caspr-coreference-tool` .

CaspR reads multiple files in CoNLL format (similar to our examples, potentially with additional columns), where each file contains annotations from one annotator. After automatic adjudication, a new CoNLL file is produced that contains either only the resulting annotation or all input annotations and the adjudication solution in separate columns of a single file for manual inspection. The result column of this file can be edited in order to enforce mentions or absence thereof, and CaspR can reprocess this file in semiautomatic mode. CaspR is realized based on Gringo (Gebser et al., 2011) and Clasp (Gebser, Kaufmann, & Schaub, 2012) version 5.

Our tool facilitates the following semiautomatic coreference adjudication process.

(1) Use CaspR to obtain a consistent adjudication of annotations with optimal usage of given information.
(2) Review the resulting CoNLL file in an editor, and enforce coreference information of certain tokens if they are clearly wrong according to the expertise of the human adjudicator. Coreference information at a token is enforced by prefixing it with the character '='.

(3) Run the re-adjudication mode of CaspR, which creates an optimal adjudication relative to manually enforced tokens.
(4) Iterate the steps (2) and (3) until a satisfactory gold standard result is obtained.

Importantly, step (3) only changes tokens that have not been enforced, hence an incremental work flow, starting with the most clear-cut cases, and ending with the most difficult cases, can be followed.

The CoNLL data format used in CaspR is variation used by the CorScorer (Pradhan et al., 2014) reference coreference resolution scoring tool. Similar to CorScorer, CaspR expects `#begin document` and `#end document` tags (which have been omitted in figures for brevity), considers the last column to be the coreference annotation column, and silently copies all other columns into the result.

For our purposes it was sufficient to use CaspR directly on CoNLL files and to use a text editor as GUI. To make the tool accessible to a wider part of the community, we consider integrating it into an existing coreference annotation toolkit, for example into BART (Versley et al., 2008).

6. Evaluation

We first describe our datasets and then experimental results.

6.1. *Datasets*

The datasets that prompted development of this application are based on the METU-Sabanci Turkish Treebank (Atalay, Oflazer, & Say, 2003; Oflazer, Say, Hakkani-Tür, & Tür, 2003; Say et al., 2004). Table 2 shows the properties of both datasets. Documents were annotated in two distinct annotation cycles: for DS1 annotators had to produce mentions and chains, while for DS2 mentions were given and could be assigned to chains or removed by annotators. This yielded a large amount of canonical mentions for DS1 (on average 316.7 per document), while DS2 contains fewer canonical mentions (159.8) although it contains by far more mentions (1561.4) than DS1. DS1 is also smaller, it is based on 21 documents from the corpus while DS2 covers the whole corpus and has more annotations per document.

In practice we observe the following: due to disagreement on canonical mention boundaries in DS1, adjudication with `[V]` or `[VA]` eliminates most data, hence using `[U]` or `[UA]` is more useful. On the other hand the 'voting' of `[V]` and `[VA]` can utilize the larger amount of annotations per canonical mention in DS1, which yields practically very usable results that do not require manual intervaention for creating a gold standard. DS1 and DS2 are structurally quite different: DS1 has several instances where nearly all mentions are (transitively) connected to all other mentions, while this does not occur in DS2.

6.2. *Experiments*

Experiments were performed on a computer with 48 GB RAM and two Intel E5-2630 CPUs (total 16 cores) using Debian 8 and at most seven concurrently running jobs, each job using at most two cores (we did not use tools in multi-threaded mode). As systems we used Gringo and Clasp from Clingo 5 (git hash 933fce) (Gebser et al., 2011; Gebser, Kaufmann, & Schaub, 2012) and Wasp 2 (git hash ec8857) (Alviano, Dodaro, Leone, & Ricca, 2015). We use Clasp with parameter `-opt-strategy=usc,9` and Wasp

Table 2. Properties of our real-world datasets which are based on the METU-Sabanci Turkish Treebank).

Dataset	DS1 (**21** instances)			DS2 (**33** instances)		
	min	avg	max	min	avg	max
# Annotators	6	6.5	8	9	10.3	11
# Chains	78	132.7	197	34	294.6	599
# Mentions	247	**596.6**	1289	117	**1561.4**	3897
# Canonical mentions	169	**316.7**	702	17	**159.8**	358
Longest chain	7	33.1	66	6	28.4	70

with parameter `-enable-disjcores`, i.e., both systems use unsat-core based optimization with stratification and disjoint core preprocessing.

Table 3 shows experimental results. We limited memory usage to 5 GB and time usage to 300 sec and we show results averaged over 5 repeated runs for each configuration. Columns MO, respectively TO, show the number of runs that aborted because of exceeding the memory limit (memory out), respectively the time limit (timeout). Columns SAT and OPT show the number of runs that found some solution and the first optimal solution, respectively, moreover we show the percentage of runs that found an optimal solution. Columns T and M give average time and memory usage which was measured with the runlim tool.[4] Columns T_{grd}, *Opt*, *Chc*, and *Cnf* show average instantiation time, optimality of the solution ($\frac{UB-LB}{LB}$), number of choices, and number of conflicts encountered in the solver. To make values comparable, columns T_{grd} through *Cnf* are accumulated only over those 44 instances where Gringo+Clasp never exceeded memory or time limits.

Overall we performed runs for 3 systems, 2 encodings, 4 objectives, and both datasets (54 instances), which yields 1296 distinct configurations. We note that memory or time limits (MO and TO) were always exceeded during solving, never during grounding.

The first accumulated results we show compare *ASP systems*, i.e., Clingo, Gringo+Clasp, and Gringo+Wasp. Wasp clearly has worse performance in this application with respect to memory as well as time compared with Clasp-based configurations. Columns T_{grd}, *Opt*, *Chc*, *Cnf* are not measurable for all runs of Wasp, therefore we omit them in these table rows. Clingo requires more memory than running Gringo+Clasp in a pipe, moreover T_{grd} of Clingo includes preprocessing time (we discuss these issues in Section 7.3). As Wasp exceeds the memory limit so often, we present only results for Gringo+Clasp in the remaining table.

In the second accumulation section of Table 3 we show a comparison between encodings CM and MM. Choosing between MM and CM means a trade-off between time and memory: while CM exceeds the memory limit less often than MM, the latter finds optimal solutions for more runs, and solutions of CM are further away from the optimum (on average 6.8) in comparison with MM (on average 1.0).

The next section of the table compares objective functions: voting-based objectives (`[V]` and `[VA]`) yield more optimal solutions in shorter time compared with the other objectives (`[U]` and `[UA]`), and even suboptimal solutions of the former objectives are often close to optimal (*Opt* is 0.0 on average for the 44 easiest instances). Moreover, strict constraints for transitive links, as used in `[U]` and `[V]`, require significantly less memory and instantiation time compared with weak constraints, which are used in `[UA]` and `[VA]` for realizing condition (C3) of Section 3.1.

The last section of the table shows practically relevant *scenarios*, accumulated over

[4]`http://fmv.jku.at/runlim/`

Table 3. Experimental results, accumulated over ASP tools, encodings, and objective functions, and accumulated over eight practically relevant use cases.

Accumulation	MO #	TO #	SAT #	OPT #	OPT %	T sec	M MB	T_{grd} sec	*Opt* avg	*Opt* max	*Chc* #	*Cnf* #
Clingo	20	0	**111**	**301**	**70**	96	738	5.7				
Gringo+Clasp	20	0	**111**	**301**	**70**	96	**695**	2.7				
Gringo+Wasp	34	81	89	228	53	145	1075	2.6				
CM	**4**	0	87	125	58	137	567	3.7	6.8	368.9	1M	331K
MM	16	0	24	**176**	**82**	54	822	**1.6**	**1.0**	**84.7**	132K	5K
`[U]`	2	0	38	68	63	125	**398**	**1.8**	1.2	15.2	2M	538K
`[V]`	2	0	3	103	95	**24**	**397**	**1.8**	0.0	0.0	177K	22K
`[UA]`	8	0	58	42	39	177	994	3.5	14.4	368.9	606K	99K
`[VA]`	8	0	12	88	81	**58**	990	3.5	0.0	0.6	132K	13K
DS1 `[U]` CM	0	0	**21**	0	0	300	384	2.8	**5.8**	**15.2**	5M	2M
DS1 `[U]` MM	2	0	1	18	86	36	806	1.0	0	0	8K	314
DS1 `[UA]` CM	2	0	19	0	0	282	1474	8.3	37.1	368.9	1M	338K
DS1 `[UA]` MM	4	0	8	9	43	136	1206	1.0	5.9	84.7	458K	38K
DS2 `[V]` CM	0	0	2	31	94	35	166	1.6	0.0	0.0	465K	69K
DS2 `[V]` MM	0	0	0	**33**	**100**	**13**	376	1.9	0	0	70K	35
DS2 `[VA]` CM	0	0	7	26	79	87	506	3.7	0.0	0.6	323K	39K
DS2 `[VA]` MM	2	0	5	26	79	64	1034	2.0	0.0	0.0	84K	134

single datasets. For DS1, non-voting-objectives are practically relevant, while for DS2 the opposite holds. DS2 can be automatically adjudicated with the MM encoding and objective `[V]` with all optimal solutions in the given time and memory. For DS1 the most feasible configuration is `[U]` with encoding CM: it never exceeds memory but unfortunately also yields no optimal solutions. In practice, having any solution is better than running out of memory. Moreover, if we increase resource limits to 8 GB and 1200 sec (not shown in the table) then we obtain optimal solutions for all documents of DS1 with `[U]` and MM, and suboptimal solutions for all documents with `[UA]` and CM.

To analyze instantiation bottlenecks, we have modified Gringo 4.5 to print the number of instantiations of each non-ground rule.[5] For an instance of average difficulty and the UA objective function, the main instantiation effort of MM encoding is the transitive closure rule (15) with 725K instantiations, while for CM it is the weak constraint (38) for transitivity with 5M instantiations. These rules clearly dominate over the next most frequently instantiated rules (12K instances for MM, 112K instances for CM). Although in encoding CM we obtain significantly more ground rules than in MM, the former requires less memory. A significant difference between the structures of CM and MM encodings is, that encoding CM is tight (Erdem & Lifschitz, 2003), while encoding MM is not, due to the transitive closure over predicate $cc/2$ in rules (13)–(15).

Note that to the best of our knowledge there are no other tools for automatic adjudication, and there are no other published datasets. Therefore we have no possibility for empirically comparing our approach with other tools or on other datasets.

[5]`https://github.com/peschue/clingo/tree/grounder-stats`

7. Discussion

We have learned the following lessons in this project.

7.1. *Approximation, Modularity, and Declarativity*

The abstract task we solve is quite straightforward. However, to make its computation feasible, we need to resort to approximations (as in assuming a maximum number of chains in the CM encoding), and we have the possibility to 'trade time for space', just as in classical algorithm development (choosing between the MM and CM encoding is such a trade-off).

Careful tuning of encodings is necessary. For example, replacing constraints (33) and (34) by the following rule and constraint

$$\begin{aligned} \mathit{clink}_{good}(M_1, M_2) &\leftarrow \mathit{result}_{cm}(C, M_1),\ \mathit{result}_{cm}(C, M_2),\ M_1 < M_2. \\ &\leftarrow \mathit{clink}(M_1, M_2),\ \textbf{not}\ \ \mathit{clink}_{good}(M_1, M_2). \end{aligned}$$

makes the encoding perform significantly worse.[6] The need for such tuning makes ASP less declarative than we would expect (or want) it to be. Still, the modularity of ASP also facilitates tuning and finding better formulations: our encodings share many rules although their essential way of representing the search space is very different.

We note, that preliminary encodings (Schüller, 2016) used different objective function formulations, however these encodings were not usable in practice without resorting to aggressive approximations that degraded results, see also Section 7.3.

7.2. *ASP Optimization*

Unsatisfiable-core-based optimization (USC) (Andres, Kaufmann, Matheis, & Schaub, 2012) and stratification (Alviano, Dodaro, Marques-Silva, & Ricca, 2015; Ansótegui, Bonet, & Levy, 2013) are both essential to the applicability of ASP in this application, and obtaining a suboptimal solution is always better than obtaining no solution at all, in particular in semi-automatic adjuciation. Branch-and-bound optimization (BB) performed so much worse than USC in this application, that we omitted any numbers.

We also experimented with additional symmetry breaking for the CM encoding, such that solutions with permutated chain IDs are prevented. With this extension of the encoding, we noticed that USC performance was reduced, while BB performance was increased (although it was not increased enough to make BB competitive with USC).

Experiments with unsat-core-shrinking of WASP (Alviano & Dodaro, 2016) have not yielded better results than with the normal WASP configuration. Similarly, experiments with lazy instantiation of constraints (Dodaro, Ricca, & Schüller, 2016; Schüller, 2016) have not yielded performance improvements.

7.3. *Instantiation Issues*

Analyzing the number of rules instantiated by non-ground rules can be useful, but it can also be misleading: in this application the encoding instantiating more constraints (CM) requires less memory in search (probably because of tightness of CM). A separate issue

[6]Due to symmetry breaking ('<') in (11) and (12), this encoding alternative is correct.

is, that using strings for input files of annotators (instead of assigning each annotator a unique integer) significantly slows down grounding and also increases memory used during grounding. Note that infeasible results discussed in (Schüller, 2016) were mostly caused by this effect. We conclude, that using strings in ASP can be a big (and non-obvious) performance issue. At the same time, it would be possible to address this issue transparently in the grounder, by converting strings to integers if no string processing operations are performed on these strings (which is not the case in our application).

Another practical issue is, that measuring instantiation time with Clingo is impossible, as Clingo reports only the sum of preprocessing and instantiation times. This makes comparisons with other systems difficult, hence we opted to compare mainly with Gringo+Clasp.

A small surprising observation is, that Clingo consistently requires slightly more memory than Gringo+Clasp.

8. Related Work

Our tool is the first automatic tool for coreference adjudication, because adjudication is usually performed manually, for example with BART (Versley et al., 2008), BRAT (Stenetorp et al., 2012), or GATE (Cunningham, Tablan, Roberts, & Bontcheva, 2013; Gaizauskas, Cunningham, Wilks, Rodgers, & Humphreys, 1996).

Our work on automatic adjudication is motivated by the amount of annotations we collected for each document, which is larger than in usual coreference annotation projects. For example the coreference annotations of the OntoNotes corpus (Hovy, Marcus, Palmer, Ramshaw, & Weischedel, 2006; Pradhan et al., 2007) were created by two independent annotators per document and adjudicated manually. Only a larger amount of annotations (we collected at least eight annotations per document) permits automatic adjudication under the assumption that the majority of annotators provides correct annotations.

The computational problem of finding minimal repairs for inconsistent annotations is related to finding minimal repairs for databases (Chomicki & Marcinkowski, 2005) or ontologies (Eiter, Fink, & Stepanova, 2014), and to managing inconsistency in multi-context systems (Eiter, Fink, Schüller, & Weinzierl, 2014). In these problems, the aim is to find a minimal change in a system such that the modified system is globally consistent, which is similar to our problem of merging mutually inconsistent coreference annotations into a single consistent gold standard result. All these problems have in common, that a change that fixes one inconsistency, might introduce another one, potentially in a part of the system that is not obviously related to the changed part.

Scoring coreference annotations based on existing and non-existing mention-mention links (as we do in the `[V]` and `[VA]` objectives) is related to the BLANC (Recasens & Hovy, 2010) evaluation measure for coreference analysis. Scoring only based on existing mention-mentin links (as in `[U]` and `[UA]` objectives) is related to the MUC (Vilain, Burger, Aberdeen, Connolly, & Hirschman, 1995) evaluation measure, which scores precision and recall of mention-mention links over all gold chains compared with all predicted chains. In our problem, we produce one set of gold chains from many other gold chains, while these scores are defined for comparing pairwise scores between two sets of chains (one the set of predicted chains, the other one the gold standard).

ASP encodings for transitivity are present in many applications. In particular ASP encodings for acyclicity properties have been studied by Gebser, Janhunen, and Rintanen (2015), including transitive closure as part of some encodings. Similar as in our experiments, tightness makes a relevant difference, but different from they do not consider optimization problems and we experiment with different ASP solvers and compare optimization algorithms.

While coreference resolution is different from coreference adjudication, methods related with Answer Set Programming have been used to perform coreference resolution. Denis and Baldridge (2009) described an approach for coreference resolution and named entity classification based on Integer Linear Programming, which includes transitivity of mention-mention links. Inoue, Ovchinnikova, Inui, and Hobbs (2012) created a approach for coreference resolution based on weighted abduction that is evaluated using an Integer Linear Programming formulation. Note that Integer Linear Programming is a formalism that is related to ASP (Liu, Janhunen, & Niemelä, 2012). Mitra and Baral (2016) describe an approach for question answering based on Inductive Logic Programming under ASP semantics, which includes coreference resolution as a subtask.

More remotely related approaches for coreference resolution include mostly methods for building coreference chains from scores on mention-mention links obtained from some machine learning solution on a labeled coreference corpus. Such methods include local greedy heuristics (Bengtson & Roth, 2008; Stoyanov & Eisner, 2012), global optimization formulations such as relaxation labeling (Sapena, Padro, & Turmo, 2012) or ranking with Markov Logic (Culotta, Wick, & Mccallum, 2007), and representations of trees of mention-mention links (Chang, Samdani, & Roth, 2013; Fernandes, dos Santos, & Milidiú, 2012). Rule-based algorithms for anaphora resolution were first described by Hobbs (1978), more recent systems merge chains in a multi-stage filtering approach (Lee et al., 2013). Hybrid systems combine rules and machine learning, and use curated or distributed knowledge sources such as WordNet, Google distance, and Wikipedia (Chen & Ng, 2012; Poesio, Mehta, Maroudas, & Hitzeman, 2004; Zheng, Vilnis, Singh, Choi, & McCallum, 2013). Most of these systems must be trained on a gold standard corpus that is the outcome of adjudication, and all of these systems are evaluated on such gold standard corpora.

9. Conclusion

We have developed an ASP application for automatic adjudication of coreference annotations along with two structurally different real-world benchmark datasets.

Adjudication is different from coreference resolution, because we do not aim to predict coreference annotations on a given text. As our work is the first automatic adjudication approach, we could not compare with systems performing the same or a similar task. Moreover, there are no standard datasets published about the problem, because usually only the final result of adjudication — the gold standard — is published.

To solve the automatic adjudication problem for all practically relevant instances that we encountered, significant effort and several iterations of encoding improvement were necessary. We have the impression, that ASP tools do not yet provide sufficient feedback about problems in encodings to make the process of encoding optimization an easy task.

Still, for our datasets, we consider this computational problem as solved, and we have integrated our encodings in the publicly available CaspR tool that supports automatic and semiautomatic adjudication of coreference data in CoNLL-format.

Future Work

Future work related to ASP in general can be the development of automatic methods for encoding optimization, similar to initial work by Buddenhagen and Lierler (2015). To be useful, such methods need to be integrated into tools, in the best case into grounders and solvers that directly give hints how to improve an encoding.

A future topic of research on coreference resolution and adjudication can be partially or largely overlapping mentions that are not annotated on exactly the same boundaries. This issue in particular arises with agglutinating languages such as Turkish where splitting long words into several tokens and annotating parts of a token can sometimes be desired. Currently, in our approach, as well as in common coreference scoring schemes, overlapping and non-overlapping mentions are treated equally as if they are completely distinct mentions.

Acknowledgements

We are grateful for the feedback received for preliminary versions of this work from anonymous reviewers at RCRA and LPNMR.

Funding

This work has been supported by The Scientific and Technological Research Council of Turkey (TUBITAK) under grant agreements 114E430 and 114E777.

References

Alviano, M., & Dodaro, C. (2016). Anytime answer set optimization via unsatisfiable core shrinking. *Theory and Practice of Logic Programming*, *16*(5–6), 533–551.

Alviano, M., Dodaro, C., Leone, N., & Ricca, F. (2015). Advances in WASP. In *International Conference on Logic Programming and Non-monotonic Reasoning (LPNMR)* (pp. 40–54).

Alviano, M., Dodaro, C., Marques-Silva, J., & Ricca, F. (2015). Optimum stable model search: algorithms and implementation. *Journal of Logic and Computation*.

Andres, B., Kaufmann, B., Matheis, O., & Schaub, T. (2012). Unsatisfiability-based optimization in clasp. In *International Conference on Logic Programming (ICLP), Technical Communications* (pp. 212–221).

Ansótegui, C., Bonet, M. L., & Levy, J. (2013). SAT-based MaxSAT algorithms. *Artificial Intelligence*, *196*, 77–105.

Atalay, N. B., Oflazer, K., & Say, B. (2003). The Annotation Process in the Turkish Treebank. In *International Workshop on Linguistically Interpreted Corpora (LINC).*

Baral, C. (2004). *Knowledge Representation, Reasoning, and Declarative Problem Solving*. Cambridge University Press.

Bengtson, E., & Roth, D. (2008). Understanding the value of features for coreference resolution. In *Empirical Methods in Natural Language Processing (EMNLP)* (pp. 294–303).

Brewka, G., Eiter, T., & Truszczynski, M. (2011). Answer set programming at a glance. *Communications of the ACM*, *54*(12).

Buddenhagen, M., & Lierler, Y. (2015). Performance Tuning in Constraint Programming. In *International Conference on Logic Programming and Non-monotonic Reasoning (LPNMR)* (pp. 186–198).

Calimeri, F., Faber, W., Gebser, M., Ianni, G., Kaminski, R., Krennwallner, T., ... Schaub, T. (2012). *ASP-Core-2 Input language format* (Tech. Rep.). ASP Standardization Working Group.

Cardie, C., Wiebe, J., Wilson, T., & Litman, D. (2004). Low-Level Annotations and Summary Representations of Opinions for Multiperspective QA. In *New Directions in Question Answering* (pp. 87–98).

Chang, K.-W., Samdani, R., & Roth, D. (2013). A Constrained Latent Variable Model for Coreference Resolution. In *Empirical Methods in Natural Language Processing (EMNLP)* (pp. 601–612).

Chen, C., & Ng, V. (2012). Combining the Best of Two Worlds: A Hybrid Approach to Multilingual Coreference Resolution. In *Empirical Methods in Natural Language Processing and Computational Natural Language Learning, Shared Task* (pp. 56–63).

Chomicki, J., & Marcinkowski, J. (2005). Minimal-change integrity maintenance using tuple deletions. *Information and Computation*, *197*(1), 90–121.

Clark, J. H., & González-Brenes, J. P. (2008). Coreference Resolution: Current Trends and Future Directions. *Language and Statistics II Literature Review.*

Culotta, A., Wick, M., & Mccallum, A. (2007). First-Order Probabilistic Models for Coreference Resolution. In *Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies* (pp. 81–88).

Cunningham, H., Tablan, V., Roberts, A., & Bontcheva, K. (2013). Getting more out of biomedical documents with GATE's full lifecycle open source text analytics. *PLoS computational biology*, *9*(2), e1002854.

Denis, P., & Baldridge, J. (2009). Global joint models for coreference resolution and named entity classification. *Procesamiento del Lenguaje Natural*, *42*, 87–96.

Dodaro, C., Ricca, F., & Schüller, P. (2016). External Propagators in WASP: Preliminary Report. In *International Workshop on Experimental Evaluation of Algorithms for Solving Problems with Combinatorial Explosion (RCRA)* (pp. 1–9).

Eiter, T., Fink, M., Schüller, P., & Weinzierl, A. (2014). Finding Explanations of Inconsistency in Multi-Context Systems. *Artificial Intelligence*, *216*, 233–274.

Eiter, T., Fink, M., & Stepanova, D. (2014). Towards Practical Deletion Repair of Inconsistent DL-programs. In *European Conference on Artificial Intelligence* (pp. 285–290).

Erdem, E., & Lifschitz, V. (2003). Tight logic programs. *Theory and Practice of Logic Programming*, *3*(4-5), 499–518.

Faber, W., Pfeifer, G., & Leone, N. (2011). Semantics and complexity of recursive aggregates in answer set programming. *Artificial Intelligence*, *175*(1), 278–298.

Fernandes, E. R., dos Santos, C. N., & Milidiú, R. L. (2012). Latent structure perceptron with feature induction for unrestricted coreference resolution. In *Joint Conference on EMNLP and CoNLL: Shared Task* (pp. 41–48).

Gaizauskas, R., Cunningham, H., Wilks, Y., Rodgers, P., & Humphreys, K. (1996). GATE: an environment to support research and development in natural language engineering. In *International Conference on Tools with Artificial Intelligence (ICTAI)* (pp. 58–66).

Gebser, M., Janhunen, T., & Rintanen, J. (2015). ASP Encodings of Acyclicity Properties. *Journal of Logic and Computation*(exv063).

Gebser, M., Kaminski, R., Kaufmann, B., & Schaub, T. (2012). *Answer Set Solving in Practice.* Morgan Claypool.

Gebser, M., Kaminski, R., König, A., & Schaub, T. (2011). Advances in gringo series 3. In *International Conference on Logic Programming and Non-monotonic Reasoning (LPNMR)* (pp. 345–351).

Gebser, M., Kaufmann, B., & Schaub, T. (2012). Conflict-driven answer set solving: From theory to practice. *Artificial Intelligence*, *187-188*, 52–89.

Gelfond, M., & Kahl, Y. (2014). *Knowledge Representation, Reasoning, and the Design of Intelligent Agents: The Answer-Set Programming Approach.* Cambridge University Press.

Gelfond, M., & Lifschitz, V. (1988). The Stable Model Semantics for Logic Programming. In *International Conference and Symposium on Logic Programming* (pp. 1070–1080).

Hirst, G. (1981). *Anaphora in Natural Language Understanding: A Survey.* Springer-Verlag.

Hobbs, J. R. (1978). Resolving Pronoun References. *Lingua*, *44*(4), 311–338.

Hovy, E., Marcus, M., Palmer, M., Ramshaw, L., & Weischedel, R. (2006). OntoNotes: the 90% solution. In *Human Language Technology Conference of the NAACL, Short Papers* (pp. 57–60).

Inoue, N., Ovchinnikova, E., Inui, K., & Hobbs, J. (2012). Coreference Resolution with ILPbased Weighted Abduction. In *International Conference on Computational Linguistics (COLING)* (pp. 1291–1308).

Kehler, A., Kertz, L., Rohde, H., & Elman, J. L. (2008). Coherence and Coreference Revisited. *Journal of Semantics*, *25*(1), 1–44.

Lee, H., Chang, A., Peirsman, Y., Chambers, N., Surdeanu, M., & Dan Jurafsky. (2013). Deterministic Coreference Resolution Based on Entity-Centric, Precision-Ranked Rules. *Computational Linguistics*, *39*(4), 885–916.

Lifschitz, V. (2008). What Is Answer Set Programming? In *AAAI Conference on Artificial Intelligence* (pp. 1594–1597).

Liu, G., Janhunen, T., & Niemelä, I. (2012). Answer set programming via mixed integer programming. In *Principles of Knowledge Representation and Reasoning (KR)* (pp. 32–42).

Mitkov, R. (1999). *Anaphora Resolution: the State of the Art.* School of Languages and European Studies, University of Wolverhampton.

Mitra, A., & Baral, C. (2016). Addressing a Question Answering Challenge by Combining Statistical Methods with Inductive Rule Learning and Reasoning. In *AAAI Conference on Artificial Intelligence.*

Mueller, E. T. (2004). Understanding script-based stories using commonsense reasoning. *Cognitive Systems Research*, *5*(4), 307–340.

Ng, V. (2010). Supervised Noun Phrase Coreference Research: The First Fifteen Years. In *Association for Computational Linguistics (ACL)* (pp. 1396–1411).

Oflazer, K., Say, B., Hakkani-Tür, D. Z., & Tür, G. (2003). Building a Turkish Treebank. In *Treebanks, Building and Using Parsed Corpora* (pp. 261–277). Springer Netherlands.

Poesio, M., Mehta, R., Maroudas, A., & Hitzeman, J. (2004). Learning to resolve bridging references. In *Annual Meeting on Association for Computational Linguistics.* (Paper #143)

Pradhan, S. S., Luo, X., Recasens, M., Hovy, E., Ng, V., & Strube, M. (2014). Scoring Coreference Partitions of Predicted Mentions: A Reference Implementation. In *Annual Meeting of the Association for Computational Linguistics (ACL)* (pp. 30–35).

Pradhan, S. S., Ramshaw, L., Weischedel, R., MacBride, J., & Micciulla, L. (2007). Unrestricted Coreference: Identifying Entities and Events in OntoNotes. In *International Conference on Semantic Computing (ICSC)* (pp. 446–453).

Recasens, M., & Hovy, E. (2010). BLANC: Implementing the Rand Index for Coreference Evaluation. *Natural Language Engineering*, *17*(4), 485–510.

Sapena, E., Padró, L., & Turmo, J. (2008). *Coreference Resolution* (Tech. Rep.). Barcelona, Spain: TALP Research Center, Universitat Politècnica de Catalunya.

Sapena, E., Padro, L., & Turmo, J. (2012). A Constraint-Based Hypergraph Partitioning Approach to Coreference Resolution. *Computational Linguistics*, *39*(4), 847–884.

Say, B., Zeyrek, D., Oflazer, K., & Özge, U. (2004). Development of a Corpus and a Treebank for Present day Written Turkish. In *Current Research in Turkish Linguistics (Proceedings of the Eleventh International Conference of Turkish Linguistics, 2002)* (pp. 182–192).

Schüller, P. (2016). Adjudication of Coreference Annotations via Finding Optimal Repairs of Equivalence Relations. In *International Workshop on Experimental Evaluation of Algorithms for Solving Problems with Combinatorial Explosion (RCRA)* (pp. 57–71).

Schüller, P. (2016). Modeling Variations of First-Order Horn Abduction in Answer Set Programming. *Fundamenta Informaticae*, *149*, 159–207.

Schüller, P. (2017). Adjudication of coreference annotations via answer set optimization. In *International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR).* (to appear)

Stenetorp, P., Pyysalo, S., Topic, G., Ohta, T., Ananiadou, S., & Tsujii, J. (2012). Brat: a Web-based Tool for NLP-Assisted Text Annotation. In *European Chapter of the Association for Computational Linguistics (EACL)* (pp. 102–107).

Stoyanov, V., & Eisner, J. (2012). Easy-first Coreference Resolution. In *International Conference on Computational Linguistics (COLING)* (pp. 2519–2534).

Tuggener, D. (2014). Coreference Resolution Evaluation for Higher Level Applications. In *European Chapter of the Association for Computational Linguistics (EACL)* (pp. 231–

235).

Versley, Y., Ponzetto, S. P., Poesio, M., Eidelman, V., Jern, A., Smith, J., … Moschitti, A. (2008). BART: A Modular Toolkit for Coreference Resolution. In *Proceedings of the ACL-08: HLT Demo Session* (pp. 9–12).

Vilain, M., Burger, J., Aberdeen, J., Connolly, D., & Hirschman, L. (1995). A Model-Theoretic Coreference Scoring Scheme. In *Message Understanding Conference (MUC)* (pp. 45–52).

Witte, R., Khamis, N., & Rilling, J. (2010). Flexible Ontology Population from Text: The OwlExporter. In *International Conference on Language Resources and Evaluation (LREC)* (pp. 3845–3850).

Zheng, J., Vilnis, L., Singh, S., Choi, J. D., & McCallum, A. (2013). Dynamic Knowledge-Base Alignment for Coreference Resolution. In *Computational Natural Language Learning (CoNLL)* (pp. 153–162).