Adjudication of Coreference Annotations via Finding Optimal Repairs of Equivalence Relations *

Peter Schüller

Computer Engineering Department
Faculty of Engineering
Marmara University, Turkey
peter.schuller@marmara.edu.tr

Abstract. We describe encodings for merging multiple coreference annotations into a single annotation, subject to hard constraints (consistency) and optimization criteria (minimal divergence from annotators) using Answer Set Programming (ASP). This task requires guessing an equivalence relation with a large number of elements. We report on experiments with real-world instances based on the METU-Sabanci Turkish Treebank using the ASP tools Gringo, Clasp, and Wasp. We also describe a patch for Gringo which facilitates fine-grained instantiation analysis and use it to analyze experimental results.

Keywords: Coreference Resolution, Adjudication, Answer Set Programming

1 Introduction

Coreference Resolution [9, 14, 20, 21] is the task of finding phrases in a text that refer to the same 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 [22] 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.*" we can identify the following mentions.

[John] $^{(i)}$ is [a musician] $^{(ii)}$. [He] $^{(iii)}$ played [a new song] $^{(iv)}$.
[A girl] $^{(v)}$ was listening to [the song] $^{(vi)}$.
"[It] $^{(vii)}$ is [[my] $^{(ix)}$ favorite] $^{(viii)}$," [John] $^{(x)}$ said to [her] $^{(xi)}$.

Roman superscripts denote mention IDs. This text contains the following chains: $\{(i), (iii), (ix), (x)\}$ (John, He, my, John); $\{(iv), (vi), (vii)\}$ (a new song, the song, It); and $\{(v), (xi)\}$ (A girl, her), where numbers refer to mention IDs.[1]

For supervised learning and verification of coreference resolution methods, annotated corpora are an important resource. For creating such resources, we usually give

* This work has been supported by Scientific and Technological Research Council of Turkey (TUBITAK) Grant 114E430.

[1] Mention $(viii)$ is in a predicative relationship with (vii). Per convention such mentions are not considered coreferent, hence $(viii)$ is missing in the second chain.

the same document to several annotators who produce mention and chain annotations independently. Afterwards we enter a process of manually adjudicating these—often conflicting—annotations to produce a single gold standard. Annotations in coreference resolution are *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. Merging such annotations is a tedious task, and the final decision on the gold standard is done based on the adjudicators' expert opinion, however tool support for this task would be useful.

We here present a method for automatic merging of coreference annotations based on combinatorial optimization, where the result combines the information provided by all annotators while performing minimal corrective modifications to ensure consistency of the output. Our method is based on ASP optimization and provides various parameters for defining which corrective modifications to prefer.

We here focus on the experimental evaluation of two encodings for this task which use different underlying representations for the equivalence relation:

- an encoding representing a transitive closure of mention-mention links, scoring repairs based on the closure, and deriving a mention-chain representation from the closure; and
- another encoding which directly represents mention-chain links and scores repairs based on pairs of mention-chain links.

We compare the Clasp [19] and Wasp [2] solver tools with branch-and-bound as well as with unsatisfiable-core optimization modes on both encodings and several objective function variations. Moreover we describe a patch to the Gringo [18] instantiation tool that allows us to shed more light on the reasons for performance of encoding variants by obtaining a detailed analysis of instantiation of nonground rules.

In the following we provide preliminaries on Coreference Resolution and Answer Set Programming in Section 2, describe our approach for automatic adjudication in Section 3, provide ASP encodings in Section 4, discuss experimental evaluation and instantiation analysis with Gringo in Section 5, and conclude with related and future work in Section 6.

2 Preliminaries

We next introduce coreference resolution informally and mathematically, and then give a brief introduction of Answer Set Programming.

2.1 Coreference Resolution

Coreference resolution is the task of finding phrases in a text that refer to the same entity [9, 14, 20, 21, 26, 33]. We call such phrases *mentions*, and we call a group of mentions that refers to one entity *chains*.

In formal mathematical terms, we can describe mention detection and coreference resolution as follows. Given a document D which is sequence of tokens $w_1, \ldots, w_n$, mention detection is the task of finding a set $M = \{(f_1, t_1), \ldots, (f_m, t_m)\}$ of mentions,

i.e., pairs (f_i, t_i) of indexes, $1 \leq i \leq m$, into the sequence of tokens ($1 \leq f_i \leq t_i \leq n$). Given a set M of mentions, coreference resolution is the task of partitioning M into k (disjoint) partitions (chains) P such that all sequences of tokens $w_{f_i}, \ldots, w_{t_i}$ corresponding to mention (f_i, t_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 $M = \{(1,1), (3,4), (6,6), \ldots, (30,30)\}$ and the correct coreference partitions 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'. □

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 [31]. In coreference annotation, humans create both mentions and chains, so the problem becomes more involved and automatic methods for creating consistent result annotations are helpful.

2.2 Answer Set Programming

Answer Set Programming (ASP) is a logic programming paradigm which is suitable for knowledge representation and finding solutions for computationally (NP-)hard problems [5, 6, 17, 23]. An ASP program consists of *rules* of the form

$$Head \leftarrow Body.$$

where $Head$ is either a first-order atom or a choice construction, and $Body$ is a conjunction of first-order atoms. Intuitively, a rule makes the head logically true if the body is satisfied. Rules without body are *facts* (the head is always true) and rules without head are *constraints* (if the body is satisfied, the candidate solution is discarded). Choices of the form $L\ \{A_1; \ldots; A_n\}\ U$ generate all solution candidates where between L and U atoms from the set $\{A_1, \ldots, A_n\}$ are true.

In addition to combinatorial search, by using *weak constraints* of the form

$$\leftsquigarrow Body. \qquad [Cost@Tuple]$$

we can perform combinatorial optimization: a weak constraint incurs cost $Cost$ for each unique tuple $Tuple$ such that $Body$ is satisfied in the answer set with respect to that tuple.

For details of syntax and semantics we refer to the ASP-Core-2 standard [7]. Efficient solvers are available for computing answer sets, in this work we use the Gringo 4.5.0 [18] grounder and experiment with the Clasp 3.1.2 [19] and Wasp 2.0 [2] solvers.

3 Automatic Coreference Adjudication

We first describe the problem more formally and give an example of inconsistent annotations, then we describe how we approach the problem and its input and output in

ASP. Finally we provide and describe two logic programs that use different internal representations.

Mathematically, we can describe adjudication of multiple coreference resolution annotations as follows.

> Given a document D of tokens as described in Section 2.1, and $u \geq 2$ partitions $P_1, \ldots, P_u$ of mentions (where each partition can be based on a different underlying set of mentions) we need to create a single set of mentions M and a single partition P which groups the set M into chains.

Clearly, annotations might be contradictory and we need to ensure certain structural constraints in the solution. For example mention A cannot be coreferent with mention B if A includes B. Moreover, a solution might contain equivalences between mentions that are not present in any annotation. This is because chains are equivalence relations, and 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 $\{(i), (iii), (ix), (x)\}$ in the Introduction, we obtain (from another annotator) a chain $\{(i), (ii), (x)\}$ If we merge these chains naively by merging the sets, we obtain a single chain $\{(i), (ii), (iii), (ix), (x)\}$ although no annotator indicated that (ii) and (iii) belong to the same chain. □

Therefore we need to merge chains in a more intelligent way.

3.1 Merge Representation and Merge Preferences

We want to use as much information from annotations as possible, while keeping consistency of the solution. We achieve this by attaching a cost to ignored mentions, to ignored links between mentions, and to links that were not annotated but arise due to merged chains.

Our conceptual model for merging annotations from multiple sources is as follows.

- A subset of annotated chain-mention links is selected from all annotators.
- From the resulting chains, each chain is represented as a set of links between all mentions it contains (i.e., as a symmetric, transitive relation).
- A subset of these links is selected to build the combined result.
- The transitive, and reflexive closure of these selected links is created, which provides an equivalence relation over mentions of all annotators.
- We use this equivalence relation as new set of chains.
- We ensure the solution adheres to two constraints: no mention can be a part of itself, and each chain must contain more than one mention.

We search for the choice of chain-mention and mention-mention links that optimizes an objective function. As objective functions we incur a cost of

- $cost_m(M)$ for each ignored mention M in a chain;
- $cost_l(M, M')$ for each ignored link between mentions M and M'; and
- $cost_a(M, M')$ for each link between mentions M and M' that is present in the result equivalence relation but not part of any annotation.

In Section 4.3 we provide concrete preference configurations of these functions.

3.2 Input and Output ASP Representation

Given u partitions $P_1, \ldots, P_u$ where each chain $C \in P_i$ contains a set of mentions of form $(from,\ to)$ we create facts of form $mention(A, M, Fr, To)$ where A represents the partition (i.e., which annotator created the annotation), M is a unique identifier for each mention, and $(Fr,\ To)$ represents the mention. Moreover we create facts of form $cm(A, Chain, M)$ for each mention with identifier M that is in a chain with identifier $Chain$ according to annotator A.

Example 3 (continued). Given the text from our running example, we can mark tokens (including punctuation) with integers as follows.

John[1] is[2] a[3] musician[4] .[5] He[6] played[7] a[8] new[9] song[10] .[11]

Consider that annotator $a1$ created chain {'John'^{a}, 'a musician'^{b}, 'He'^{c}} while annotator $a2$ produced {'musician'^{d}, 'He'^{e}} where superscripts indicate mention IDs. These annotations would be represented by the following ASP facts.

$$\begin{array}{ll} mention(a1, a, 1, 1) \leftarrow. & cm(a1, c1, a) \leftarrow. \\ mention(a1, b, 3, 4) \leftarrow. & cm(a1, c1, b) \leftarrow. \\ mention(a1, c, 6, 6) \leftarrow. & cm(a1, c1, c) \leftarrow. \\ mention(a2, d, 4, 4) \leftarrow. & cm(a2, c2, d) \leftarrow. \\ mention(a2, e, 6, 6) \leftarrow. & cm(a2, c2, e) \leftarrow. \end{array}$$

where $c1$ and $c2$ represent the first and second chain, respectively. □

The output of our logic program will be a set of chains without annotator information. Therefore we want to obtain atoms of the form $result_{cm}(Chain, mid(Fr,\ To))$ which indicate that in chain $Chain$ there is a mention from token Fr to token To.

Example 4 (continued). Assume we have merged the annotations of the previous example into two chains $v = \{$'John', 'a musician'$\}$ and $w = \{$'musician', 'He'$\}$, this would be represented by the following atoms:

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

Note that it is frequently the case that mentions contain other mentions, but this is usually forbidden in case these mentions belong to the same chain. □

4 ASP Encodings

We next provide two ASP encodings that realize the above merging strategy. The first encoding explicitly represents the transitive closure of mention-mention links, while the second encoding avoids such a representation by representing the resulting equivalence relation only as chain-mention links. We use functions $cost_X$ for configuring the preference relation; these are part of the encoding and resolved during instantiation.

$$\{\ use_{cm}(A, Chain, M) : cm(A, Chain, M)\ \} \leftarrow . \tag{1}$$
$$\leftsquigarrow cm(A, C, M), \mathbf{not}\ use_{cm}(A, C, M). \qquad [cost_m(M)@1, A, C, M, cm] \tag{2}$$
$$link(A, M_1, M_2) \leftarrow use_{cm}(A, C, M_1), use_{cm}(A, C, M_2),\ M_1 < M_2. \tag{3}$$
$$\{\ use_{link}(A, M_1, M_2)\ \} \leftarrow link(A, M_1, M_2). \tag{4}$$
$$\leftsquigarrow link(A, M_1, M_2), \mathbf{not}\ use_{link}(A, M_1, M_2). \quad [cost_l(M_1, M_2)@1, A, M_1, M_2, mm] \tag{5}$$
$$clink(X, Y) \leftarrow X = mid(Fr_1, To_1), Y = mid(Fr_2, To_2), X < Y,$$
$$use_{link}(A, M_1, M_2), mention(A, M_1, Fr_1, To_1), mention(A, M_2, Fr_2, To_2). \tag{6}$$
$$clink(X, Y) \leftarrow X = mid(Fr_1, To_1), Y = mid(Fr_2, To_2), X < Y,$$
$$use_{link}(A, M_2, M_1), mention(A, M_1, Fr_1, To_1), mention(A, M_2, Fr_2, To_2). \tag{7}$$
$$clink^{\rightarrow}(X, Y) \leftarrow clink(X, Y). \tag{8}$$
$$clink^{\rightarrow}(X, Y) \leftarrow clink^{\rightarrow}(Y, X). \tag{9}$$
$$clink^{\rightarrow}(X, Z) \leftarrow clink^{\rightarrow}(X, Y), clink^{\rightarrow}(Y, Z). \tag{10}$$
$$\leftsquigarrow clink^{\rightarrow}(X, Y), X < Y, \mathbf{not}\ clink(X, Y). \quad [cost_a(X,Y)@1, X, Y, add] \tag{11}$$
$$not_{repr}(Y) \leftarrow clink^{\rightarrow}(X,Y), X < Y. \tag{12}$$
$$result_{c,m}(X, Y) \leftarrow clink^{\rightarrow}(X, Y), \mathbf{not}\ not_{repr}(X). \tag{13}$$
$$result_c(C) \leftarrow result_{c,m}(C, \_). \tag{14}$$
$$\leftarrow result_c(C),\ \#count\ \{\ Y : result_{c,m}(C, Y)\ \} \leq 1. \tag{15}$$
$$\leftarrow clink(mid(F_1, T_1), mid(F_2, T_2)), F_1 \leq F_2, T_2 \leq T_1, (F_1, T_1) \neq (F_2, T_2). \tag{16}$$

Fig. 1. Encoding variant MM (full).

4.1 Mention-Mention Encoding (MM)

Figure 1 shows encoding MM. Rule (1) guesses which subset of mention-chain annotations from all annotators is used, while the weak constraint (2) incurs cost $cost_m(M)$ for each mention-chain annotation that is not present in the solution. Each pair of distinct mentions in the same chain induces a link between these mentions (3), rules (4) guess a subset of these links to be used in the result, and the weak constraint (5) incurs a cost $cost_l(M, M')$ for each link that is not present in the solution. Note that we canonicalize the symmetric $link$ relation using the $<$ operator based on link IDs.

Rules (6) and (7) project away annotator information and use new constant terms of form $mid(Fr, To)$ to identify links between mentions using just start and end tokens Fr and To, respectively.[2] The relation $clink(\cdot, \cdot)$ is a full canonical representation of all mentions and mention-mention links in a candidate solution.

Rules (8)–(10) define in $clink^{\rightarrow}$ the symmetric, reflexive, and transitive closure of the $clink$ relation. This closure represents an equivalence relation between mentions, i.e., the result chains, and is the distinguishing feature of the MM encoding.

[2] Both rules are required because IDs of mentions could be sorted differently from their occurrence in the text.

$$ann(A) \leftarrow cm(A, \_, \_). \tag{17}$$
$$count\_chain(A, N) \leftarrow ann(A), N = \#count\ \{\ C : cm(A, C, \_)\ \}. \tag{18}$$
$$maxchain(N * 3/2) \leftarrow N = \#max\ \{\ C : count\_chain(\_, C)\ \}. \tag{19}$$
$$\{\ result_c(X) : X = 1..Max\ \} \leftarrow maxhain(Max). \tag{20}$$
$$result_c(X-1) \leftarrow result_c(X), 1 < X. \tag{21}$$
$$cmention(M) \leftarrow clink(M, \_). \tag{22}$$
$$cmention(M) \leftarrow clink(\_, M). \tag{23}$$
$$1\ \{\ result_{cm}(C, M) : result_c(C)\ \}\ 1 \leftarrow cmention(M). \tag{24}$$
$$\leftarrow clink(X, Y), result_{cm}(C, X), \mathbf{not}\ result_{cm}(C, Y). \tag{25}$$
$$\leftarrow clink(X, Y), \mathbf{not}\ result_{cm}(C, X), result_{cm}(C, Y). \tag{26}$$
$$\leftsquigarrow result_{cm}(C, X), result_{cm}(C, Y), X < Y, \mathbf{not}\ clink(X, Y). \quad [cost_a(X, Y)@1, X, Y, add] \tag{27}$$

Fig. 2. ASP rules for encoding variation CM which replaces rules (8)–(11).

Based on $clink^{\rightarrow}$, weak constraint (11) incurs cost $cost_a(M, M')$ for all pairs of mentions that are present in the result equivalence relation but not in any mention-mention link obtained from annotators ($clink$).

The equivalence relation $clink^{\rightarrow}$ is transformed back into a chain-mention representation in (13) and (14) by using the lexicographically smallest mention as representative for each chain, where (12) identifies lexicographically non-smallest mentions.

Finally we impose two structural constraints on the result: a chain has to contain more than one mention (15) and a mention that is within another mention cannot be in the same chain with that mention (16). Note that, thanks to the elaboration tolerance [25] of ASP, constraints (15) and (16) can be removed if the respective restriction should not be applied, or can be made more restrictive (e.g., mentions in a chain cannot be neighbors) independent from the rest of the encoding.

The encoding shown in Figure 1 admits as answer sets all possible consistent merges of the given annotations, including solutions where all annotations are ignored. Weak constraints prefer solutions where, according to the chosen $cost$ functions, as many given annotations as possible are used and a single consistent solution is represented in the answer set. The balance between not ignoring annotations and not creating too many novel links between mentions can be adjusted using the $cost$ functions.

4.2 Chain-Mention Encoding (CM)

The difference between MM and CM is the way we represent the equivalence relation and how we check the preference function, in particular links in the solution that are not present in annotations. For reasons of brevity, we provide this encoding as a patch to encoding MM: encoding CM is obtained by using rules (17)–(27) shown in Figure 2 instead of rules (8)–(11) in encoding MM.

In the CM encoding variant, (17) represents annotators, (18) finds the number of chains for each annotator, and (19) computes 1.5 times the maximum of the largest number of chains annotated by any annotator in the input. Encoding CM is based on the assumption that the preferred result will not contain more chains than this amount of chains. This assumption can safely be made because differences in the amount of annotations are consistent over the whole document: some annotators produce more annotations, but this is true over the whole document, therefore using the maximum amount of chains times 1.5 is safe (using the preference relations we study, this number is never reached in practical solutions).

Given the maximum number of resulting chains, (20) guesses which chain will actually exist, and (21) ensures that if we use a certain chain ID, then we also use the ID below (this eliminates symmetries by ensuring the usage of consecutive chain IDs). Rules (22) and (23) represent all canonical mentions in $cmention(\cdot)$. For each of these mentions, (24) guesses in which chain that mention will be in the result. Constraints (25) and (26) enforce, that the guessed solution in $result_{cm}$ corresponds to those links of annotators that have been selected to be part of the solution in $clink$.

To incur a cost for transitive edges that are not present in annotations, we cannot use $clink^{\rightarrow}$ as in encoding MM. Instead, in weak constraint (27) we again join the $result_{cm}$ relation with itself on the chain ID.

4.3 Weak and Strong Constraints for Preferences

The encodings we have described so far impose a cost on chain/mention and mention/mention annotations that are not reflected in the solution, and a cost on mention/mention annotations in the solution that were not annotated by human annotators.

If we replace the weak constraints by strict constraints (and remove the attached cost) then we obtain solutions corresponding to infinite cost of a constraint violation, i.e., the constraints must never be violated. Note that, if we replace all weak constraints by strict constraints, inconsistent annotations will not admit a solution. To guarantee existence of a solution, we must allow either to ignore chain/mention annotations, i.e., $cost_m(M)$ must be finite, or to ignore mention/mention annotations, i.e., $cost_l(M, M')$ must be finite.

Table 3 shows all objective function configurations we experiment with in this work. A dash '-' indicates that the constraint was used as a strict constraint, an integer indicates a constant cost per ignored annotation, while L indicates a cost depending on the length $|M|$ of mention M, computed as follows.

$$cost_m(M) = \begin{cases} 3 \text{ if } |M| = 1 \\ 2 \text{ if } 2 \leq |M| \leq 4 \\ 1 \text{ otherwise} \end{cases} \qquad cost_l(M, M') = \begin{cases} 3 \text{ if } |M| + |M'| = 2 \\ 2 \text{ if } 3 \leq |M| + |M'| \leq 8 \\ 1 \text{ otherwise} \end{cases}$$

Some rationale for the choice of objectives in Table 3 is as follows. Annotators make more mistakes with longer mentions, and for this reason it can be useful to assign a higher cost to ignoring mentions that contain fewer tokens. Therefore cost functions

Encoding/Approximation	OOM #	OOT #	SAT #	OPT #	T sec	M MB	T_{grd} sec
CM/25	0	182	592	234	239	498	4.0
CM/∞	96	245	562	105	250	1372	10.1
MM/25	181	150	324	353	175	1329	10.8
MM/∞	181	162	354	311	186	1339	10.9

Table 1. Variation of encoding variant and approximation.

can be constant or depend on the length of the involved mentions. We can ensure transitivity by enforcing that we never merge chains that would require additional mention-mention links, however this would cause more mentions or their links to be ignored, and we would like to believe that human annotators create annotations for a reason. Therefore we put a cost on additional links that is lower than the cost of ignoring a given annotation (in some cases the cost is even zero). Moreover ignoring a chain-mention link implicitly removes at least one mention-mention link, and in large chains removes many mention-mention links. Therefore we put higher cost on ignoreing chain-mention links than on ignoring mention-mention links. A more comprehensive analysis of the implications of cost functions in Table 3 for the practical usefulness of solutions is a topic of future work. In this work our purpose is to provide a configurable framework and study the influence on the cost function on feasibility of computation.

5 Experimental Evaluation

To evaluate our approach we used 21 documents from the METU-Sabanci Turkish Treebank [29] where documents have between 494 and 2600 mentions (1199 on average) and between 247 and 1300 chains (600 on average) with each chain containing between 19 and 53 mentions (34 on average). Each document was annotated by between 6 and 9 annotators (8 on average).

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 8 concurrently running jobs (each job using a single core, as we did not use solvers in multithreaded mode). We limited memory usage to 5 GB and time usage to 300 sec. For instantiation we used Gringo 4.5.0 [18], for solving Clasp 3.1.2 [19] and Wasp 2.0 [2]. We use Clasp with parameters `-opt-strategy=bb` and `-opt-strategy=usc` and Wasp with parameters `-weakconstraints-algorithm=basic` and without arguments (which corresponds to unsat-core based optimization).

As our instances are very hard for the abovementioned memory and time constraints, we additionally experiment with an approximation for the preference relation: we only use a subset of weak constraints, concretely we omit weak constraints (for ignoring or adding mention-mention links) whenever the pair of mentions has a distance of more than 25 tokens in the document.

In our experimental results, columns OOM, OOT, SAT, and OPT, shows the number of runs that exceeded 5 GB, exceeded 300 sec, found at least one satisfiable solution,

Encoding/Solver	OOM #	OOT #	SAT #	OPT #	T sec	M MB	T_{grd} sec	Chc #	$Conf$ #	Lem #
CM/Clasp/B	0	0	241	11	291	402	4.1	8M	1M	1M
CM/Wasp/B	0	0	247	5	297	1039	4.0	1M	41M	297K
CM/Clasp/U	0	120	0	132	156	248	4.0	1M	492K	492K
CM/Wasp/U	0	62	104	86	213	305	4.0	1M	2M	31K
MM/Clasp/B	42	0	199	11	269	1158	10.8	3M	1M	1M
MM/Wasp/B	49	83	112	8	251	1641	10.9	514K	11M	295K
MM/Clasp/U	42	27	0	183	79	1055	10.8	316K	10K	10K
MM/Wasp/U	48	40	13	151	102	1462	10.7	114K	478K	2K

Table 2. Variation of computation method and encodings with 25-token approximation.

and found the first optimal solution and proved its optimality, respectively. Moreover columns T, M, and T_{grd} give the average total time, memory, and instantiation time of all runs. In our experiments, all OOM and OOT conditions were reached during solving, never during grounding.

For comparing the feasibility of computing solutions with MM and CM encodings and with or without approximation, Table 1 depicts accumulated results over all 21 documents, 4 solver variations, and 12 objective functions (i.e., each table row summarizes 1008 runs). We can observe that encoding MM exhausts memory much more than encoding CM, and that we can only fit into 5 GB by using CM and the 25-token approximation. Although CM fits into less memory, it requires more time on average and we find more optimal solutions with encoding MM, both with and without the 25-token approximation. Instantiation times are similar for all configurations, except for the CM/25 combination which requires half the time of the other configurations for reasons we try to explain in Section 5.1.

Table 2 compares computational methods for both encodings with 25-token approximation across all 21 instances and 12 objective functions. We see that for getting some answer set (SAT or OPT), bound-based optimization and CM encoding are better than configurations with unsat-core optimization or encoding MM, i.e., CM/·/B provides (suboptimal) solutions for all instances. However, the highest number of optimal solutions is obtained with encoding MM and unsat-core optimization: here Clasp performs better (183 optimal solutions) than Wasp (151) in this setting. The columns Chc, $Conf$, and Lem, show the number of choices, conflicts, and learned clauses, respectively (in runs that did not exhaust memory). Clasp did not find any satisfiable solutions in unsat-core (U) mode (either the solutions were optimal or the timeout was reached), while Wasp found many solutions before optimality. On the other hand, in branch-and-bound mode (B) Clasp finds more optimal and more satisfiable solutions than Wasp. The memory usage of Wasp is significantly higher than the one of Clasp in mode B, while for CM/·/U this difference is smaller (248 MB vs 305 MB).

In summary, for branch-and-bound optimization Clasp seems more suitable, while for unsat-core optimization, Wasp seems more suitable.

As only the CM encoding with 25-token approximation fits into memory for all configurations, and because only branch-and-bound optimization finds solutions in all

Objective			Clasp/B				Wasp/B			
$cost_m$	$cost_l$	$cost_a$	SAT #	OPT #	T sec	M MB	SAT #	OPT #	T sec	M MB
-	1	-	21	0	300	560	21	0	300	804
-	1	0	19	2	274	443	20	1	287	1132
1	-	-	21	0	300	547	21	0	300	1981
1	-	0	19	2	274	161	20	1	292	1686
2	-	1	20	1	295	190	21	0	300	1383
2	1	-	21	0	300	567	21	0	300	706
2	1	0	19	2	281	520	20	1	295	727
3	2	1	21	0	300	366	21	0	300	888
6	3	1	20	1	294	423	21	0	300	857
-	L	1	20	1	287	450	20	1	288	806
L	-	1	20	1	288	194	20	1	291	770
2L	L	1	20	1	292	402	21	0	300	721

Table 3. Variation of objective with encoding CM, branch-and-bound, 25-token approximation.

cases, we compared several possible objective functions using the CM/25 configuration. Table 3 shows the result, both with Clasp and Wasp in branch-and-bound optimization mode. We see that there are no big differences between the feasibility of various objective functions: only few instances are solved to the optimal solution and therefore average time requirements are very close to the timeout of 300 seconds. However, we can observe interesting memory requirement, which is high for Wasp in those cases where it is low for Clasp: for an objective function which does not permit ignoring mention-mention links, uses fixed cost 2 on ignoring chain-mention annotations, and either uses cost 1 or no cost for additional mention-mention links. Moreover, memory requirement is more stable for Clasp while it varies more and is higher in general for Wasp.

5.1 Instantiation Analysis with Gringo

To gain additional insights about the instantiated program, we have modified Gringo 4.5.0 to count instantiations of each nonground rule and extend the output of `gringo -verbose` with a table that shows for each nonground rule the number of times it was instantiated with 0 body conditions (i.e., as facts), with 1 and 2 body conditions (i.e., as rules that can be propagated particularly efficiently), and as rules with 3 or more body conditions. Our publicly available patch[3] shows these counts for the intermediate representation of Gringo.

Table 4 shows those parts of the instantiation analysis table where rules have been instantiated more than 10k times for an instance of average difficulty (1298 mentions, 649 chains, max 35 mentions per chain).

In the MM encoding, the main instantiation effort (725k rules) originates in the transitive closure rules (10). On the other hand, in the CM encoding the main instantiation

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

Variables in body				Nonground Rule
0	1	2	3+	MM encoding
0	12K	0	0	`clink_tr(X,Y):-clink_tr(Y,X).`
0	0	725K	0	`clink_tr(X,Z):-clink_tr(Y,Z),clink_tr(X,Y).`
0	453	12K	0	`result_cm(X,Y):-clink_tr(X,Y),not not_repr(X).`
0	12K	0	0	`result_c(C):-result_cm(C,_).`
0	1	2	3+	CM encoding
0	0	0	112K	`:-result_cm(C,X),clink(X,Y),not result_cm(C,Y).`
0	0	0	112K	`:-result_cm(C,Y),clink(X,Y),not result_cm(C,X).`
0	0	5212K	112K	`:~result_cm(C,Y),X<Y,result_cm(C,X), not clink(X,Y).`

Table 4. Nonground rules with more than 10k instantiations in an instance of average difficulty, as provided by patched Gringo tool.

effort (5212k rules) is the weak constraint that incurs a cost on mention-mention links which are not present in any annotation (27). All other rules are instantiated less than 10k times, so the shown rules dominate over other rules in both cases.

Two interesting things become apparent when relating the times in Table 2 and the numbers in Table 4: although encoding CM contains significantly more instantiated rules than MM (5M compared with 800K) (a) encoding CM can be instantiated much faster than MM (4.1 sec compared with 11.1 sec), moreover (b) encoding CM requires less memory and produces fewer OOM conditions than MM.

A significant difference in the structure of CM and MM encodings is, that CM is tight [15], i.e., it requires no unfounded-loop check and can be solved by solving a SAT instance, while encoding MM is not tight due to the transitive closure rule.

As a result of these observations, it appears as if loop nogoods produced by solvers in encoding MM cause the observed OOM conditions and the increased memory footprint. Still, encoding MM permits to find more optimal solutions, which indicates that — if there is sufficient memory available — the non-tight encoding MM is to be preferred over the tight encoding CM.

6 Conclusion

We have developed a method for automatic merging of coreference resolution annotations, with the objective of using as much information from annotators as possible while maintaining consistency of the resulting annotation.

In practice, adjudication cannot be done automatically, therefore our approach allows to enforce a part of the resulting mentions and chains, while optimizing a solution over the remaining tokens that were not (yet) specified by the user.

We have created a tool based on our encodings and based on the popular CoNLL format for coreference resolution and the format used by the CorScorer [27] tool. We

are currently using this method to prepare a corpus for Turkish coreference annotations and plan to release the tool as open source software in the future.

6.1 Related and Future Work

ASP encodings for transitivity are present in many applications. In particular ASP encodings for acyclicity properties have been studied by Gebser et al. [16] including transitive closure as part of some encodings. Similar as in our experiments, tightness makes a relevant difference. Different from Gebser et al. we consider optimization problems and compare different ASP solvers and different optimization algorithsm.

Finding minimal repairs for inconsistent annotations is related to finding minimal repairs for databases [8] or ontologies [13], and to inconsistency management in distributed knowledge based systems [12]. In these related applications, the aim is to find a minimal change in the system that make it globally consistent, as in the merging of coreference annotations. All these applications have in common, that a change which fixes one inconsistency might introduce another one, as in the case of coreference annotations, where adding a mention-mention link can implicitly create many other invalid mention-mention links, and removing a mention-mention link can require us to remove many mention-mention links that have been annotated.

Maximizing the usage of annotations provided by annotators based on mention-mention links is similar to the MUC [32], B^3 [4], and BLANC [28] evaluation metrics for coreference analysis which are based on mention-mention links. In particular the most recently proposed BLANC metric puts an emphasis on including non-existing mention-mention links between gold standard and system output into the score, which is related to our constraints on mention-mention links that are not present in annotations. More remotely related to our work is the CEAF [24] metric which is based on finding an optimal bipartite graph matching between system output and gold annotations, and identifying percentage of either correct chains or correct mentions. Realizing an optimization criterion based on this idea in our work would be an interesting topic for future work.

Denis and Baldridge described an approach for coreference resolution and named entity classification based on Integer Linear Programming [10], which is a formalism related to Answer Set Programming. Their approach is not aimed at merging annotations and does not support a semi-automatic operation, however they include in their formulation a similar constraint on transitivity of mention-mention links as we have in rules (9) and (10).

Our current encodings are not yet feasible for large inputs which occur in practice, as shown in our experiments. In many cases the optimal solution is not found and only with approximation of the objective function we can find some solution for all instances within 5 minutes. In future work we will study the impact of optimality on the usability of a solution, and we will investigate the choice of preference function for making our method useful for adjudication in practice. To increase performance, we plan to apply recently developed solver versions that support stratification [3] and unsat-core-shrinking [1] for providing suboptimal answer sets while using unsat-core methods, moreover we plan to apply methods for lazy instantiation of constraints [11, 30].

References

1. Alviano, M., Dodaro, C.: Anytime answer set optimization via unsatisfiable core shrinking. Theory and Practice of Logic Programming pp. ICLP 2016, In press, arXiv:1608.00731 [cs.LO] (2016)
2. Alviano, M., Dodaro, C., Leone, N., Ricca, F.: Advances in WASP. In: International Conference on Logic Programming and Non-monotonic Reasoning (LPNMR). pp. 40–54 (2015)
3. Ansótegui, C., Bonet, M.L., Levy, J.: SAT-based MaxSAT algorithms. Artificial Intelligence 196, 77–105 (2013)
4. Bagga, A., Baldwin, B.: Algorithms for Scoring Coreference Chains Shortcomings of the MUC-6 Algorithm. In: International conference on language resources and evaluation workshop on linguistics coreference. pp. 563–566 (1998)
5. Baral, C.: Knowledge Representation, Reasoning, and Declarative Problem Solving. Cambridge University Press (2004)
6. Brewka, G., Eiter, T., Truszczynski, M.: Answer set programming at a glance. Communications of the ACM 54(12) (2011)
7. Calimeri, F., Faber, W., Gebser, M., Ianni, G., Kaminski, R., Krennwallner, T., Leone, N., Ricca, F., Schaub, T.: ASP-Core-2 Input language format. Tech. rep., ASP Standardization Working Group (2012)
8. Chomicki, J., Marcinkowski, J.: Minimal-change integrity maintenance using tuple deletions. Information and Computation 197(1), 90–121 (2005)
9. Clark, J.H., González-Brenes, J.P.: Coreference Resolution: Current Trends and Future Directions. Language and Statistics II Literature Review (2008)
10. Denis, P., Baldridge, J.: Global joint models for coreference resolution and named entity classification. Procesamiento del Lenguaje Natural 42, 87–96 (2009)
11. Dodaro, C., Ricca, F., Schüller, P.: External Propagators in WASP: Preliminary report. In: International Workshop on Experimental Evaluation of Algorithms for Solving Problems with Combinatorial Explosion (RCRA) (2016)
12. Eiter, T., Fink, M., Schüller, P., Weinzierl, A.: Finding Explanations of Inconsistency in Multi-Context Systems. Artificial Intelligence 216, 233–274 (2014)
13. Eiter, T., Fink, M., Stepanova, D.: Towards Practical Deletion Repair of Inconsistent DL-programs. In: European Conference on Artificial Intelligence. pp. 285–290 (2014)
14. Elango, P.: Coreference Resolution: A Survey. Tech. rep., University of Wisconsin, Madison (2005)
15. Erdem, E., Lifschitz, V.: Tight logic programs. Theory and Practice of Logic Programming 3(4–5), 499–518 (2003)
16. Gebser, M., Janhunen, T., Rintanen, J.: ASP Encodings of Acyclicity Properties. Journal of Logic and Computation (2015)
17. Gebser, M., Kaminski, R., Kaufmann, B., Schaub, T.: Answer Set Solving in Practice. Morgan Claypool (2012)
18. Gebser, M., Kaminski, R., König, A., Schaub, T.: Advances in gringo series 3. In: International Conference on Logic Programming and Non-monotonic Reasoning (LPNMR). pp. 345–351 (2011)
19. Gebser, M., Kaufmann, B., Schaub, T.: Conflict-driven answer set solving: From theory to practice. Artificial Intelligence 187–188, 52–89 (2012)
20. Hirst, G.: Anaphora in Natural Language Understanding: A Survey. Springer-Verlag (1981)
21. Kehler, A., Kertz, L., Rohde, H., Elman, J.L.: Coherence and Coreference Revisited. Journal of Semantics 25(1), 1–44 (2008)
22. Lee, H., Chang, A., Peirsman, Y., Chambers, N., Surdeanu, M., Dan Jurafsky: Deterministic Coreference Resolution Based on Entity-Centric, Precision-Ranked Rules. Computational Linguistics 39(4), 885–916 (2013)

23. Lifschitz, V.: What Is Answer Set Programming? In: AAAI Conference on Artificial Intelligence. pp. 1594–1597 (2008)
24. Luo, X.: On coreference resolution performance metrics. In: Conference on Human Language Technology and Empirical Methods in Natural Language Processing (HLT/EMNLP). pp. 25–32 (2005)
25. McCarthy, J.: Elaboration tolerance. In: Common Sense. pp. 1–19 (1998)
26. Mitkov, R.: Anaphora Resolution: the State of the Art. School of Languages and European Studies, University of Wolverhampton (1999)
27. Pradhan, S., Luo, X., Recasens, M., Hovy, E., Ng, V., Strube, M.: Scoring Coreference Partitions of Predicted Mentions: A Reference Implementation. In: Annual Meeting of the Association for Computational Linguistics (ACL). vol. 2, pp. 30–35 (2014)
28. Recasens, M., Hovy, E.: BLANC: Implementing the Rand Index for Coreference Evaluation. Natural Language Engineering 17(4), 485–510 (2010)
29. Say, B., Zeyrek, D., Oflazer, K., Özge, U.: Development of a Corpus and a Treebank for Present Day Written Turkish. In: (Proceedings of the Eleventh International Conference of Turkish Linguistics, 2002) Current Research in Turkish Linguistics, pp. 182–192. Eastern Mediterranean University Press (2004)
30. Schüller, P.: Modeling Variations of First-Order Horn Abduction in Answer Set Programming. Fundamenta Informaticae (2016), to appear. arXiv:1512.08899 [cs.AI]
31. Stoyanov, V., Eisner, J.: Easy-first Coreference Resolution. In: International Conference on Computational Linguistics (COLING). pp. 2519–2534 (2012)
32. Vilain, M., Burger, J., Aberdeen, J., Connolly, D., Hirschman, L.: A Model-Theoretic Coreference Scoring Scheme. In: Message Understanding Conference (MUC-6). pp. 45–52. ACL Anthology (1995)
33. Zheng, J., Chapman, W., Crowley, R., Savova, G.: Coreference resolution: A review of general methodologies and applications in the clinical domain. Journal of Biomedical Informatics 44(6), 1113–1122 (2011)