RAMFIS System Report TAC 2018

Cecilia Mauceri, Shafiuddin Rehan Ahmed, and Timothy O'Gorman
University of Colorado Boulder

I. Introduction

As TA2 performers, our directive is to meld knowledge elements resulting from TA1s multi-modal extractions from individual documents into a single knowledge base where that information is integrated into a single structure, and contradictions and confirmations can be recognized. For the TAC 2018 evaluation, we focused on cross-document co-reference linking, as shown in Figure 1. In this document, we will go into detail on our architecture, linking procedure, and challenges we faced during the evaluation.

II. Jena Database

As we are integrating information into a very large triple store of facts, we rely upon Apache Jena[1] – in particular, TDB2 – to handle queries into the knowledge base. In the current (non-streaming) form, we load all data into a single database, and develop clusters using queries into the entire KB. Since we need the information from the entire data before clustering, it wasnt practical nor possible to load it into memory. TDB2 provides an in-file based solution to handle this situation. It also does some indexing on the triples that gives a reasonable performance on the entire data. However, for the purpose of linking, we extracted all the items of each class into serialized forms outside of Jena to avoid querying TDB2 on each run and running operations such as partial string matching through SPARQL.

III. Consuming TA1 Output

Documents were loaded into the database by running Jena TDB2Loader on all individual TA1 documents to form a single knowledge base. Minor formatting issues in TA1 inputs were addressed with simple scripts run over those files, but little pre-processing was done otherwise. The resulting KB contained a great deal of redundancy, and certain justifications asserted by TA1s were not actually justified in the text – we estimate that if we had planned to develop methods to remove redundant information from the TA1 inputs, it could have dramatically reduced the size of the resultant KB.

Some formatting was done to make inputs match the strict assumptions of the NIST SPARQL queries. As the TA1s we worked with were not using those queries, their format did not match specific assumptions – in particular, they largely eschewed use of SameAsCluster representations for clustering, and did not represent edge clusters using CompoundJustification assertions. We added this to the KB to make these assertions match the NIST assumptions.

IV. Linking

Linking is performed in three steps. First, we cluster together all entities which TA1s have already linked to existing entries in the background knowledge base – a step which provided a large portion of our links, for the TA1s which provided such information. Second, we did heuristic-driven entity linking (discussed in the entity linking section) to link the remaining entities using name matching across documents, and to link entities across TA1s through justification matching. Finally, we used those entity clusters to make event linking decisions.

To simplify the problem, we used an overarching constraint of linking entities and events of the same type. This serves to improve performance in two ways, both by partitioning the entity linking problem so that there are fewer pairwise comparisons, and by enabling parallelization of the problem. This filters out obvious non-links, although there are natural edge cases (as in the distinction between GPE and LOC) where it can be overly rigid.

We also made named-based entity linking decisions in a pairwise manner based upon local information. This entails that the coreference clustering occurs in a greedy manner, which can naturally result in overly large clusters and propagate error, as large linked clusters have an even greater tendency to add more links. Therefore for this evaluation, we choose very conservative comparison functions, valuing precision over recall.

Links are the only changes we made to knowledge elements. We consumed the entities and events generated by TA1 and added co-reference links, but did not otherwise modify the extractions, other than changes to formatting to match the NIST restrictions to the AIF format.

A. Entity Linking

Entity resolution varied greatly depending upon whether the TA1 team provided entity linking information connecting their entities to the background knowledge base (LinkAssertion statements). Because the TA1 teams had so much more information available in the surface text, we made two strong assumptions – both that we trusted the links provided by TA1s, and trusted that if they did not link an otherwise valid entity to that KB entry, they were correct in doing so. In practice, that means that when processing information from the ISI team, we focused upon nil clustering, clustering together the entities not linked to the external knowledge base.

All such entities without external KB links were partitioned by their type (e.g. person, geopolitical, etc) and the

Fig. 1. System Overview. This flow chart illustrates the process of adding TA1 output to the knowledge base and performing cross-document co-reference linking. We perform linking in two steps. First we link entities, followed by events. The event linker uses links found by the entity linker in addition to other evidence.

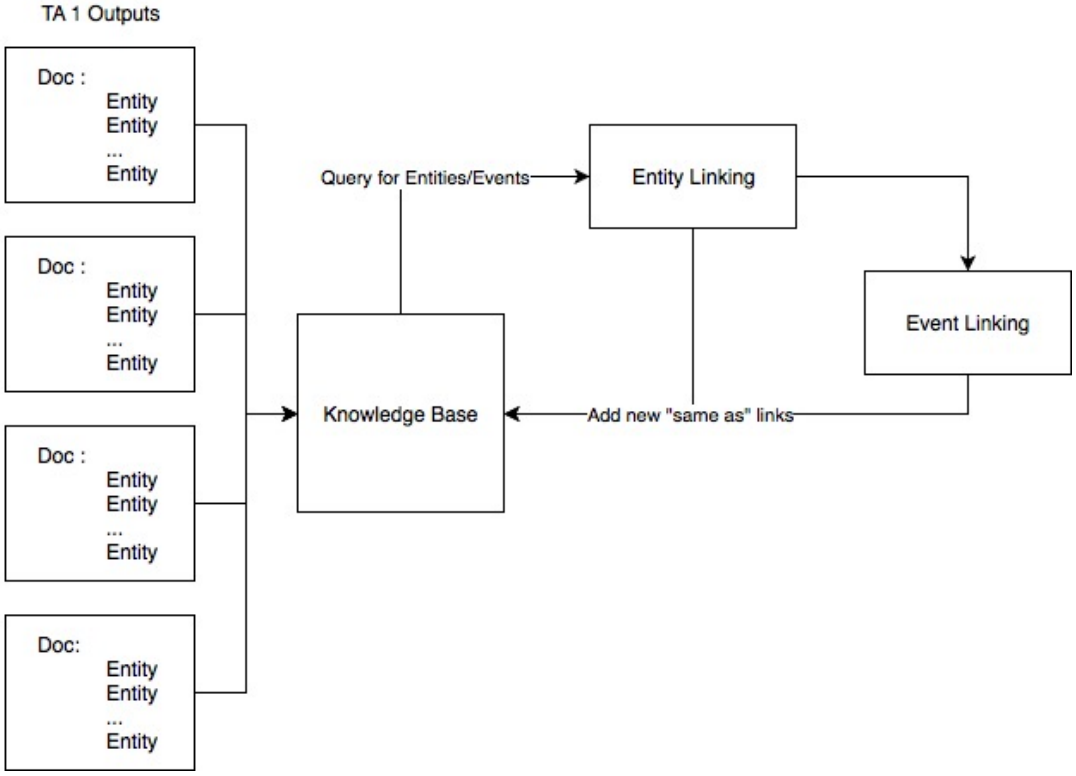
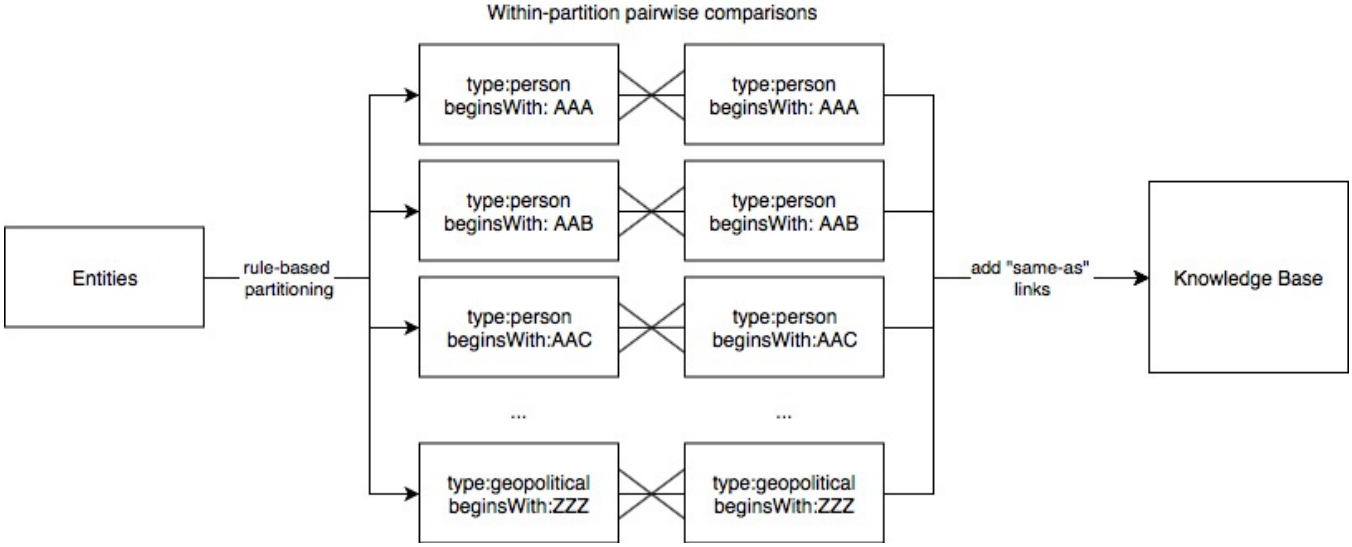


Fig. 2. Details of a Linking Module. This flowchart shows the partition, compare, link pipeline. In this example, partitions are defined by entity type and the first three characters of the entity name.



first three characters of their name string. For the TAC 2018 evaluation, the largest such partition was 8600, and the average partition size was about 40.

The entity comparison function is rule-based and focused upon name comparison. This uses a series of progressively more lenient conditions, starting with exact match linking, and backing off to partial match. Entities are also represented by a context feature, which represents the names of entities mentioned in proximity to each entity within a document. This allows us to measure whether entities share relationships to other entities, even when that relation is not explicitly captured in the KB.

With partial string matching, context similarity measure, and a fixed threshold, we build an adjacency list of the entities for each partition. Using the adjacency list, we find the connected components within the partitions to give us clusters of coreferent entities. So entities with names Barack and Obama, although are different string-wise, get clustered together because they are both connected to the entity Barack Obama. Also, a John with a context of John Smith does not get clustered with a John with a context of John Doe. We then proceed to add the sameAsClusters links into the KB where the first entity of the cluster is the canonical entity and all the other entities are cluster members. We also make sure the cluster prototypes provided by the TA1s do not end up becoming cluster members. In other words, a cluster prototype is preferred over the first entity of the cluster to be the canonical entity.

B. Event Linking

Although we developed supervised models trained on some existing event data (EventCorefBank), we ended up using rule-based methods for event linking in the pilot, due to the scale of the pilot evaluation data being dealt with. We focused upon a simple entity linking heuristic based upon shared event arguments. In this approach, after the entity linking pass was applied across the entire KB, for each entity that had the same event-argument relation to two events of the same event type, we simply assumed that that pair of events was coreferent. Coreference clusters were determined greedily by a sequence of those pairwise decisions. Section 5: Querying the Database Our output KB was very large. As individual TA1s primarily passed us data using hard clustering assertions (locating multiple justifications under a single entity or event), our TA2 entity and event resolution was primarily additional information. We therefore output a KB that was fundamentally a union of all the TA1 documents, merged with the added set of SameAsClusters provided by our entity and event linking components.

Having the large output documents as TDB databases, we then ran NIST SPARQL queries, as provided in the xml files for classes, zero-hop queries and graph queries. Although NIST provided scripts for parsing KBs, those tools seemed to be explicitly written only for TA1 performers, both because they loaded KBs in memory (not practical with our KB sizes) and because of the inefficient SPARQL query issues noted below. We were limited in coverage of some graph representations, therefore, as NIST only provided SPARQL format for the simplest, one hop graph queries (it is quite possible that this pilot system would not have found matches with the more complex, multi-hop graph queries).

In practice, we concluded that the provided NIST queries were not designed to be applied at scale, and we needed to write scripts to re-format and optimize the queries. Our re-ordering code structured the queries to start by finding entry points – which were either hasName assertions or combinations of source documents and justification start and end points – and to then add links from there. This allowed us to run the SPARQL queries on our large (30+ gb) knowledge base.

Scripts were then written to convert those outputs to the NIST xml output format, and validation of those xmls against the provided Validation toolkit. This was complicated by the absence of any documentation for the format; we primarily relied upon an example output nestled within the NIST query toolkit.

V. Challenges and Workarounds

1) The primary challenge was that as an independent TA2, we did not have any prior knowledge of the expected input. We therefore were surprised by the size of the incoming KB. Having more examples of the expected inputs and outputs at each stage of the pipeline would have dramatically helped in giving us more accurate representations of the task.
2) The flexibility of the AIF was compounded by the offloading of query processing to each team. In practice, we felt that there was no forcing function requiring a consistent output between teams.
3) The intractibility of NIST SPARQL queries resulted in many wasted hours both within our team and related teams we worked with. The NIST xmls and output format were also both undocumented, which hindered our ability to develop our own tools. Good prior warning about the NIST formats would have made our work much more efficient.
4) The sheer size of the resultant KB was a challenge, and it resulted in complications in even simple tasks (such as writing the KB to disk).
5) Query efficiency played a large role in the run time of our system. We found out that the order in which we write the SPARQL statements was particularly crucial. For example, if we want to get the type of an entity, a query such as:

```
?statement ?type rdf:Statement .
?statement rdf:predicate rdf:type .
?statement rdf:subject ?entity_id .
?statement rdf:object ?entity_type .
```

Is way slower than:

```
?statement rdf:subject ?entity_id .
?statement rdf:predicate rdf:type .
?statement ?type rdf:Statement .
```

```
?statement rdf:object ?entity_type .
```

REFERENCES

[1] Carroll, J. J., Dickinson, I., Dollin, C., Reynolds, D., Seaborne, A., & Wilkinson, K. (2004, May). Jena: implementing the semantic web recommendations. In Proceedings of the 13th International World Wide Web Conference (pp. 74-83). ACM.