KB Construction and Hypothesis Generation Using SAMSON

Carl Andersen[1], Drew Wicke[1], Kyle Tunis[1], Wheeler Howard[1], Mark Gerken[2], Dustin Carroll[2], Cassidy Harless[2], Cecilia Newell[2] and Theresa Swift[3]

[1]Raytheon BBN Technologies - {carl.andersen,drew.n.wicke,kyle.h.tunis,wheeler.howard}@raytheon.com
[2]Polaris Alpha - {mark.gerken,dustin.carroll,cassidy.harless,cecilia.newell}@polarisalpha.com
[3]Coherent Knowledge, LLC - theresasturn@gmail.com.

Abstract

This document describes SAMSON, Raytheon BBN and partners' system used in the KB Construction and Hypothesis Generation tasks of the 2019 TAC SM-KBP competition. For KB Construction, SAMSON performs entity linking and disambiguation using vector embeddings of the KB. For Hypothesis Generation, SAMSON uses two different methods of relaxed querying. The first method constructs spaces of hierarchical clusters over various classes in the KB. We then translate both the KB and the query to cluster-level semantics and use a Prolog meta-interpreter to run the query. The second method uses local graph search to find KB nodes similar to those requested in the query.

1 Introduction

This document describes the Semantic Abduction over Multi-Source ObservatioNs (SAMSON) system and associated research from the Raytheon BBN Intelligence Exploitation (IX) group, partner Polaris Alpha, and consultant Coherent Knowledge. The SAMSON team entered our system in the 2019 Text Analysis Conference (TAC) Streaming Multimedia Knowledge Base Population (SM-KBP) track. The long-running Text Analysis Conference (TAC) emphasizes empirical evaluation of competing systems in various subfields of Natural Language Understanding. SM-KBP also serves as an evaluation exercise for performers on the DARPA Active Interpretation of Disparate Alternatives (AIDA) program. SM-KBP explores the problem of extracting content from large numbers (>1000) of multimedia (text, pictures, video, speech) documents and producing a unified knowledge base (KB) in which recurring entities and events are co-referred. The track also explores the problem of hypothesis generation, in which multiple, mutually contradictory *hypotheses* (narratives) explaining some event are identified in the unified KB.

The SAMSON team and system participated in two of the task areas (TAs) for SM-KBP 2019: TA2 (KB Construction) and TA3 (Hypothesis Generation). Our SAMSON system computes rigorous solutions over datasets of over 100M triples in only a few hours for TA2 and less than an hour for TA3. We describe our participation in each TA in separate sections. Each section describes its task in more detail, then presents the research and design of the relevant part of the SAMSON system, and finally, offers a brief discussion of experimental results.

2 Architecture

Figure 1 shows the SAMSON architecture and data flow for TA2 and TA3. SAMSON uses a Java+Spring front end API that also mediates between its core reasoning engines. These include the XSB Prolog reasoner [24], used for TA3 reasoning and TA2/TA3 semantic translation; an array of Python/C data and embedding tools used in TA2; the Polaris Alpha Mentieta Miner engine, used in TA3; and the Lucene text indexing system, used in TA3.

3 TA2: KB Construction

To represent semantic data in all three task areas, SM-KBP uses the AIDA Interchange Format (AIF), a semantic ontology developed for the AIDA program and written in OWL. TA2 (KB Construction), assumes that TA1 (Extraction) has already processed a large number (>1000) of multimedia documents, extracting and outputting a semantic data file in AIF for each input document.

This research was developed with funding from the Defense Advanced Research Projects Agency (DARPA). The views, opinions and/or findings expressed are those of the author and should not be interpreted as representing the official views or policies of the Department of Defense or the U.S. Government. Distribution Statement "A" (Approved for Public Release, Distribution Unlimited)

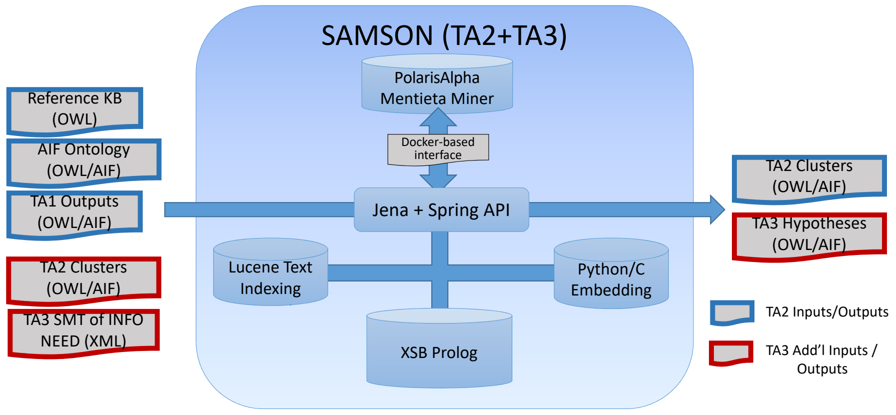


Figure 1: SAMSON high level architecture.

TA2 comprises two activities. First, TA2 performs entity disambiguation (co-reference) between Knowledge Elements (KEs), the Entity, Relation, and Event (ERE) instances in the TA1 output files. Second, where possible, TA2 performs entity linking, i.e. links Entity instances with counterparts in an AIF *Reference KB* developed by the Linguistic Data Consortium (LDC). By computing co-references and linkages, TA2 effectively integrates all the TA1 outputs and the Reference KB into a single KB, which we refer to as the *Unified KB*.

SAMSON's TA2 pipeline uses state of the art Python-based graph embedding to perform entity linking and co-reference resolution of multilingual text. XSB Prolog is used at various points in the pipeline to perform semantic translations of the AIF inputs and outputs. Embedding approaches map data of various kinds to high-dimensional metric spaces; the computed embedding supports generalized similarity queries used to identify likely co-reference and linking relationships.

3.1 TA2 Related Work

Several approaches perform entity disambiguation and linking using embeddings [20, 9]. However, these approaches process textual documents instead of semantic graphs derived from the documents, as we do.

More similar to our problem and methods are approaches that produce embeddings of network structures, such as Attri2Vec [27], RDF2Vec [22], and DeepWalk [19]. RDF2Vec processes RDF graphs, producing embeddings of the graph nodes that predict existing edges. RDF2Vec predicts only edges incident upon a node instead of edges in larger neighborhoods. In contrast, Deepwalk uses unlabeled graphs and attempts to predict target nodes obtained from random graph walks starting at an input node. Deepwalk represents graphs using sparse vectors, making it incompatible with dense vector inputs such as we obtained from fastText. Attri2Vec predicts randomly selected nearby nodes like Deep Walk and uses expressive dense vector inputs representing node attributes (e.g. name, type), but does not use node neighborhoods in the input as SAMSON does.

Our system is novel with respect to all the above work because we incorporate both node attributes and neighborhoods into the input vector. This produces an embedding space that is sensitive to both these features, making it possible to, for example, differentiate among two instances of an event type by their slot values.

3.2 TA2 System Overview

The core of our TA2 system is a pipeline of open-source, Python-based toolsets for data processing, embedding, and clustering. These include:

- Pandas: tabular data structures and analysis tools [14]
- Apache Arrow/Parquet: a compressed, efficient columnar data representation easily used in pandas
- NetworkX: a library for representing and manipulating complex attributed networks.
- numpy: an efficient tool for manipulating and storing multidimensional array data [15]
- FAISS: a GPU-accelerated similarity search and clustering of dense vectors (at a billion-scale) [12]
- Scikit-learn: a library of efficient machine learning algorithms; we use the DBSCAN [6] and OPTICS [1] unsupervised clustering algorithms[18]
- PyTorch: a GPU accelerated deep neural networks and tensor computations library [17]
- fastText: pre-trained embedding models for multiple languages [3]. fastText uses subword information to produce vectors for out-of-vocabulary (oov) words

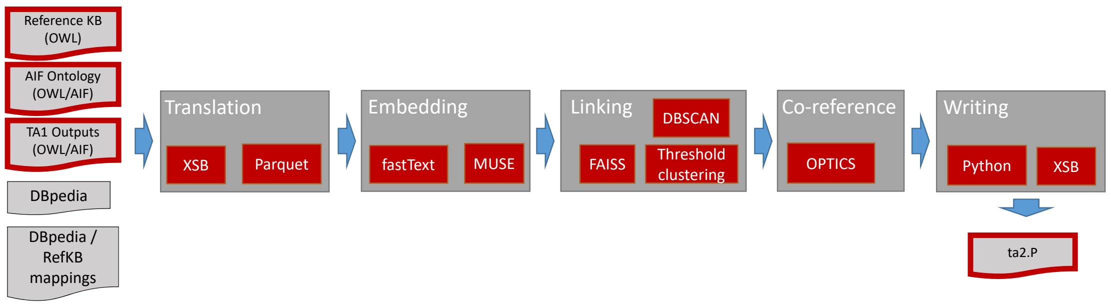


Figure 2: SAMSON TA2 data flow.

- MUSE: a library for multilingual word embeddings based on fastText [4]

Figure 2 shows our TA2 pipeline. At the left, we engineer AIF/Prolog (.P) data files that link Entities from the Reference KB with a reduced version of DBpedia [2], thereby enriching these entities with additional properties exploited by the embedding process. The Reference KB's origin as a translation of Wikipedia to OWL makes linking it with DBpedia a straightforward process of ID matching. To ensure consistency with SM-KBP data, we also translate relevant DBpedia classes to counterpart classes in AIF; these class conversions are identifed by hand. This enriched Reference KB is converted to Parquet for easier Python-based processing.

In parallel, XSB converts TA1 AIF data to Prolog (.P) format, which can be processed by both XSB and Python. The python pipeline reads in the .P files and converts to Parquet for easier Python-based processing.

Next, the enriched Reference KB and the TA1 data are separately processed by our graph embedding pipeline. Our embedding scheme uses textual values derived from ERE names and classes as inputs. To capture the shared meaning of linguistic word and phrasal homonyms over different languages, we convert textual values to embeddings using the fastText and MUSE systems. Because the Reference KB is in English, we use fastText embeddings; we use MUSE to convert the multi-language TA1 data to English embedding spaces.

We also add contextual meaning of the node in the graph by performing a random walk of length l (in our experiments $l = 4$) starting with the desired node and averaging the fastText representations of each of the nodes in the path. We repeat the random walk k times (in our experiments $k = 3$) and we averaged the resulting vectors. This process produces a 300 length vector that represents the context of the node and is then appended to the fastText representation of the node itself (which also has a length of 300). The node representation then is a 600 length vector.

The resulting vector representations of the entities, events, and relations are clustered using FAISS, DBSCAN, and OPTICS. Entities are linked to the reduced reference KB by using threshold based clustering. The resulting clusters are then written out to a Prolog based representation and then transformed into AIF using XSB.

3.3 FastText and MUSE

FastText provides pre-trained models for many languages allowing us to produce embeddings for out-of-vocabulary words that we use as input for our own embedding framework. These word embedding encode semantics of words (i.e. Kramatorsk and Kramatorsk Airport nearby in vector space). This enables our system to handle noisy data.

There are a number of problems associated with fastText. For example, fastText produces null (all zeros) embeddings for words with character length <2. Therefore, for short abbreviations such as US and UK, we convert fastText inputs by hand, (e.g. we convert 'US' to 'United States of America').

In addition, misspelled words are not always embedded in a region similar to the correct spelled word. Another issue we encountered is due to the way data is passed between TA1 and TA2, there is no way to incorporate the context in which a given word was seen. For example, the sentence that the word is found.

One disadvantage (common to all discussed) is that fastText only produces a vector for words of length greater than 2. If the string is 2 or less the vector is all zeros. Therefore, for missing and small strings we end up with a vector of zeros. Therefore, in some cases such as abbreviations for the United States of America as in US we convert to United States of America.

FastText does provide pre-trained models for a number of languages, however these models are not aligned. Therefore, in order to produce vectors for multiple languages aligned in a single vector space we must use a method like MUSE. MUSE aligns the vector spaces in order to produce a new vector in the target space. We used MUSE to align the fastText vector spaces of Russian and Ukrainian to English.

By aligning the spaces we have representations of words in the three languages closer together. An example of English and Russian alignment is shown in Figure 4.

3.4 Data Ingest

The first stage in our pipeline is data ingest. We first convert TA1 AIF data into XSB readable .P files which we then convert to attribute graphs using Python's pandas and NetworkX libraries. An attribute graph, a built-in NetworkX data structure, is an unlabeled graph comprised of nodes/edges that have named attributes that have some data such as a string or vector.

We ingest all TA1 data files to produce one XSB readable Prolog file containing one Prolog statement for each TA1 AIF statement. In order to reduce the size and ease the processing of TA1 data, we convert it from reified, multiple statement form to a de-reified form using XSB. Ingested and transformed Prolog are shown in Figure 4.

In addition to the TA1 data, we also ingest the Phase 1 Reference KB, reduce its size, and enrich it with DBpedia. The Phase 1 Reference KB contains a very large number of entities (in the millions) and a large proportion of them are not relevant, mainly due to the inclusion of all entities in the GeoNames [25] dataset. In order to reduce the reference KB's size we merge the GeoNames references found in the Reference KB with DBpedia entities that have GeoNames URIs. This merge reduces the Reference KB from >10M statements to several hundred thousand statements. To support use of the DBpedia entitues, we manually align DBpedia's ontology with AIF. DBpedia has approximately 767 classes and AIF has 372. Figure 5 shows a sample alignment.

3.5 Embedding Approaches

We developed three different embedding systems: a *baseline* approach which relied only on fastText vectors, a *network embedding* approach which uses a neural network to learn a vector representation based on fastText input, and a *graph embedding* approach based in part on the previous two approaches.

Baseline. Our baseline embedding system leverages public word embedding models, particularly fastText, which supports words in multiple languages and similarity queries for out of vocabulary words using morpheme embeddings. Our prototype system embeds TA1 KEs based on their textual values (hasName, textValue, numericValue). For each textual value, we obtain a fastText embedding used as the final embedding for the node. This baseline approach obviously is only as robust as fastText, and often is inaccurate for homographs (e.g. the same fastText vector is produced for the soda Coca-Cola and the movie called Coca-Cola).

Network Embedding. We also completed a prototype of a more robust attributed network embedding system that uses both node attributes and node network (neighborhood) information. Our system is based on Attri2Vec, a neural network algorithm that learns an embedding for nodes that contain attribute features, such as the fastText vectors of text or type information. Attri2Vec incorporates both attribute information and network structure in its learned vector representation and, like DeepWalk, uses a learning objective that predicts neighboring nodes.

We first convert our input RDF graph to an attribute graph. For each node, we create one or more attributes by obtaining fastText vectors for the node's type name and/or textual value. The attributes for a node and for its neighbors are combined via averaging and concatenation to obtain an overall input value for the node. Given a node input of this

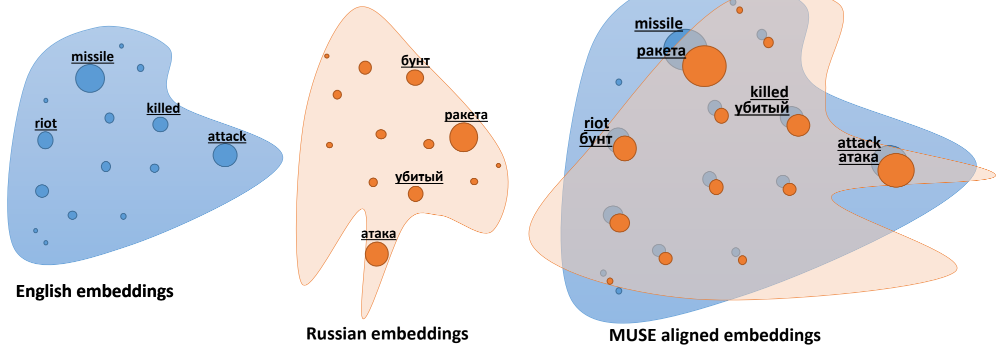
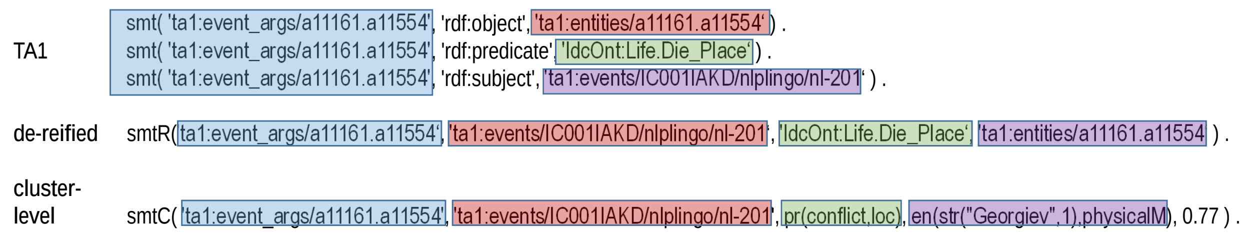


Figure 3: MUSE aligning embedding space into a common space.



```
TA1            smt( 'ta1:event_args/a11161.a11554', 'rdf:object', 'ta1:entities/a11161.a11554' ) .
               smt( 'ta1:event_args/a11161.a11554', 'rdf:predicate', 'ldcOnt:Life.Die_Place' ) .
               smt( 'ta1:event_args/a11161.a11554', 'rdf:subject', 'ta1:events/IC001IAKD/nlplingo/nl-201' ) .

de-reified     smtR('ta1:event_args/a11161.a11554', 'ta1:events/IC001IAKD/nlplingo/nl-201', 'ldcOnt:Life.Die_Place', 'ta1:entities/a11161.a11554' ) .

cluster-
level          smtC( 'ta1:event_args/a11161.a11554', 'ta1:events/IC001IAKD/nlplingo/nl-201', pr(conflict,loc), en(str("Georgiev",1),physicalM), 0.77 ) .
```

Figure 4: Transformation of a reified TA1 edge (in AIF) to de-reified and cluster-level representations. The colored boxes track the evolution of individual edge elements. The cluster-level representation is used in TA3; its last element is a confidence value.

form, the neural network learns to predict the occurrence of neighboring and nearby nodes chosen via random walks. The learned neural layer immediately prior to the output represents the node embedding. More formally we have the following:

$$f(x_i) = \mathrm{ReLu}(x_i W^{in}) W^{out}$$

where x_i is the fastText feature vector of node v_i in the graph, W^{in} are the input weights, W^{out} is an embedding layer. We train using a negative log loss function using the Adam optimizer in PyTorch. We perform negative sampling by selecting $K+1$ output weights where the first index is the neighboring node and the other K are indices corresponding to noise nodes as used in [27]. The objective is to predict the neighboring node and once trained we are able to produce vector representations by using the input weights W^{in}.

Graph Embedding. Our current implementation of the Network Embedding approach requires very long training time (>12 hours) for the sizable SM-KBP TA1 data inputs. Furthermore, to complete even these runs, even using modern GPUs, we are forced to reduce the learned embedding vector size, resulting in poor performance in which node context is largely lost. Therefore, for the SM-KBP evaluation, rather than attempting to learn a node embedding representation, we use the input into the network embedding neural network *as* the representation. This enables us to both retain the contextual information of the node and improve the speed of obtaining the embedding for each node. However, this representation only imperfectly expresses the semantic differences between nodes.

3.6 Clustering

Density based spatial clustering of applications with noise (DBSCAN) is a clustering technique that clusters KEs based on a user supplied density threshold. Figure 6 presents a simple illustration of the DBSCAN concept in two dimensional space. Each of the KEs (dots, triangles, squares) are mapped into this 2-D space. Pairwise distances are computed using the afficient FAISS algorithm. A KE (named p) is chosen and the number of points within a defined distance threshold, ϵ, are counted. If there are more than a minPoints threshold KEs around p and within distance ϵ, then p is a core point (dot). All KEs within ϵ are clustered into a single cluster. Multiple KEs can be core points within this cluster. Any other KEs (triangles) within ϵ of any core point in a cluster are included in the cluster. KEs that are further away than ϵ from a core point are not included and if further away than ϵ from all core points then they are

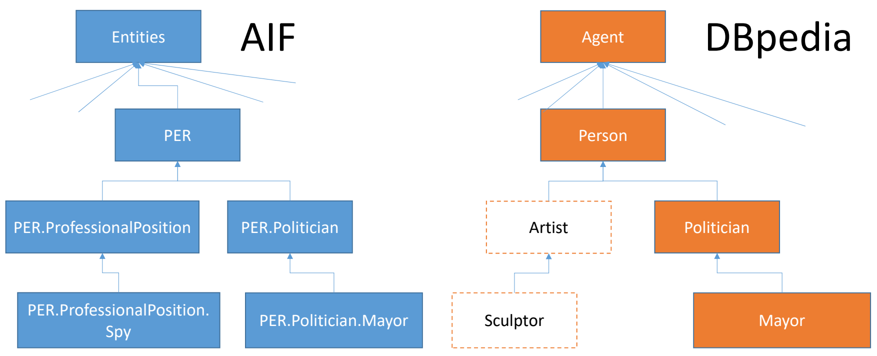


Figure 5: Example alignment of AIF and DBpedia classes. DBpedia's Artist class has no direct counterpart in AIF, so it is made a subclass of AIF PER.

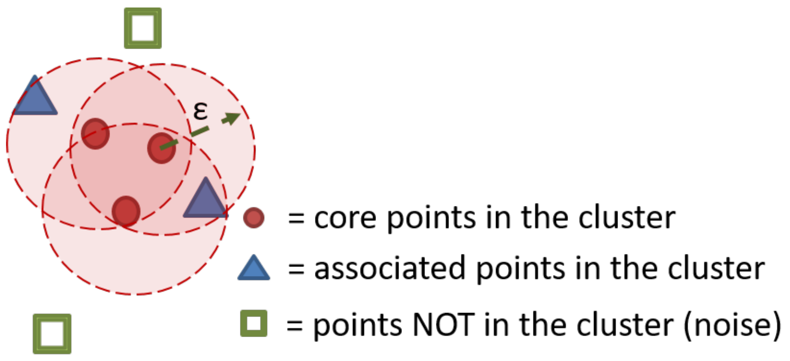

ε
= core points in the cluster
= associated points in the cluster
= points NOT in the cluster (noise)

Figure 6: KE clustering in DBSCAN based on the minimum distance threshold and a minimum number of points threshold. Red dots are core points indicating that they meet or exceed the minPoints threshold within the distance threshold.

considered as noise (squares). This process iterated until all KEs belong to either a named cluster or are assigned to the noise cluster.

Ordering points to identify the clustering structure (OPTICS) is a clustering algorithm that relaxes the density requirement of DBSCAN. The basic idea is that by relaxing the density requirement, relevant small clusters can be assigned that would be otherwise missed in using density based thresholds. In OPTICS, distances between all points are calculated. The KEs are then reordered so that spatially close points are neighbors after the ordering. We use a minPts-based threshold, i.e. the core distance (ϵ) of point p is determined by the smallest distance such that p meets the minPts threshold. The reachability distance between a point q and p is the minimum of either the core distance, ϵ, or the distance from p to q. This process iterates until all points are processed. Clusters are formed from sets of points with shorter distances.

3.7 Co-Reference

Entity linking is performed between a reduced Phase1Eval KB and TA1 entities using FAISS (fast distance calculation) to perform threshold based clustering. The entities that fall outside the threshold are not linked to a reference KB entity and thus become part of a NIL Cluster. We create NIL (non-linked) Clusters using DBSCAN with entities that fall outside the threshold. We create singleton clusters from DBSCAN noise entities. This process is illustrated in Figure 7.

Unlike entities, events and relations are not linked to a reference KB, and so do not require threshold based clustering. Instead, we use OPTICS to cluster

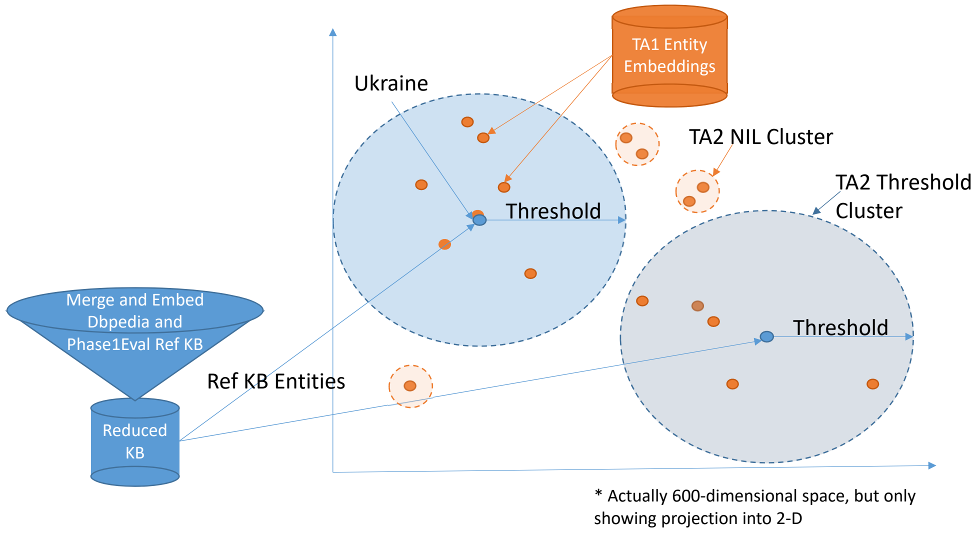

TA1 Entity Embeddings
Ukraine
TA2 NIL Cluster
TA2 Threshold Cluster
Threshold
Threshold
Merge and Embed Dbpedia and Phase1Eval Ref KB
Ref KB Entities
Reduced KB
* Actually 600-dimensional space, but only showing projection into 2-D

Figure 7: Linking and entity disambiguation using the embedding space.

co-referent events and relations.

3.8 TAC 2017 V-measure Results

We tested our SAMSON-TA2 using the V-measure [23] cluster quality metric. V-measure measures the similarity between a gold standard cluster set and a test cluster set. For the gold standard set, we used the TAC 2017 TA2 clusters published by LDC; we ran TAC 2017 TA1 outputs provided by the BBN TA1 team through our system to obtain a test cluster set.

Because V-measure assumes that the gold and test clusters operate over identical instance sets, we first had to identify a common set of Entity instances found in both the BBN TA1 and LDC gold data. For this purpose, we assumed that Entities from the two datasets that had common types and overlapping textual justifications were the same Entity instance. There are 24,373 Entity instances in the LDC gold dataset and 7,524 in the BBN dataset. Using the matching method described above, only 1,875 Entity instances (7.693% of the gold instances) occur in both datasets; we ran the V-measure metric using only these instances and their associated clusters. We obtained a V-measure score of 0.98008 with a homogeneity of 0.99957 and a completeness of 0.96134. Therefore, 98% of the found Entity elements were correctly co-referenced. However, the vast majority of these were singleton clusters.

3.9 TA2 Future Work

We have a number of future work headings to improve our embedding based TA2 approach. We intend to address one of the failings of fastText by incorporating MOE (Misspelling Oblivious (word) Embeddings) to handle misspelled words. Also, in order to improve our clustering results we intend on incorporating automatic parameter tuning and more formal analysis of our choice in clustering algorithms. This will be a great improvement as our current approach is to hand tune the parameters in order to reduce runtime and increase the number of clusters generated.

Additionally, we will leverage sentence embeddings to produce KE embeddings. In particular, we hope to experiment with BERT/XLNet as a replacement of fastText [5, 26]. Our new approach would convert random walks through TA1 graph into a sentence, which can then be queried using BERT/XLNet for context sensitive word embeddings that represent the KE element in the graph.

In addition to improving our KE embeddings we hope to use our embeddings to predict missing information. To do so we will use BERT's masked language model to predict missing words/entities in the sentence constructed from a graph. We will seek to speed up training by performing transfer learning using a pre-trained BERT model along with a subset of TA1 data.

4 TA3: Hypothesis Generation

4.1 SINs and Hypotheses

TA3, Hypothesis Generation, assumes TA2 has already produced the Unified KB and attempts to identify multiple hypotheses in the Unified KB that answer a given Statement of Information Need (SIN), or semantic query. In keeping with the Semantic Web representation used by SM-KBP, a SIN is represented as a set of entry points and edges somewhat similar to a SPARQL [21] query; Figure 8 shows a portion of a SIN used in SM-KBP. Like a SPARQL triple pattern, each SIN entry point or edge contains one or more variables. One query answer (hypothesis) finds ground versions of these edges in the Unified KB, implicitly binding edge/entry variables to nodes in the Unified KB.

However, a hypothesis answer has several differences from a SPARQL answer. First, SPARQL represents answers as sets of bound variables that implicitly represent edges in a data graph. In contrast, a hypothesis answer simply returns the data graph edges themselves. Second, SPARQL has a rigid logical semantics that requires all variables to be bound consistently and all data edges to match the given triple pattern. In contrast, a hypothesis answer allows arbitrary relaxation of the answer edge, such that the answer edge may not match the SIN edge at all! This flexibility is required because error-prone

```
<!-- Mykola Serhiyenko (a.k.a. Nikolai Sergienko) -->
 <entrypoint>
    <node> ?DeceasedSerhiyenko </node>
    <typed_descriptors>
      <typed_descriptor>
        <enttype> PER.ProfessionalPosition </enttype>
        <kb_descriptor>
          <kbid> LDC2019E43:80000112 </kbid>
        </kb_descriptor>
      </typed_descriptor>
    </typed_descriptors>
  </entrypoint>
```

```
<!-- Who carried out the killing of Mykola Serhiyenko? -->
  <edge id="AIDA_M18_TA3_E101_F1_E1">
    <subject> ?DeathOfSerhiyenko </subject>
    <predicate> Life.Die.DeathCausedByViolentEvents_Victim </predicate>
    <object> ?DeceasedSerhiyenko </object>
  </edge>
  <edge id="AIDA_M18_TA3_E101_F1_E2">
    <subject> ?DeathOfSerhiyenko </subject>
    <predicate> Life.Die.DeathCausedByViolentEvents_Killer </predicate>
    <object> ?KillerOfSerhiyenko </object>
  </edge>
```

Figure 8: Selected entrypoint and edges from a Statement of Information Need (SIN).

TA1/TA2 linguistic processing (e.g. typing errors, missed extractions) may produce a noisy unified KB whose edges only partially match the SIN edge.

The final difference between hypothesis answers and SPARQL answers is that the hypothesis must also include a set of coreference clusters and Reference KB linkages for each node included in an answer edge. Ideally, the cluster/linkage requirement would allow hypotheses to express that the same entities and edges were encountered in multiple input documents, allowing a user to trace the hypothesis back to all the documents that support it. However, because of the difficulty of gathering and co-referring all possible supporting evidence in formats acceptable to NIST, our team simply made singleton clusters for this supportive evidence, effectively returning only one of potentially many instances of the hypothesis pattern in the supporting data.

SIN entry points are a special case of the above matching process. The function of entry points is to bind a single variable found in some SIN edge, thereby constraining the hypothesis answers that should be returned. While several different forms of entry point were specified by NIST, Phase 1 used only Reference KB entry points, so we limit our discussion to these. A Reference KB entry point consists of an edge variable, a name string, and an ontology class. To process this entry point, a TA3 system should find some node in the Reference KB whose name matches the given string and whose class matches the given class; the variable is then bound to the node. As with all other matching functions in TA3, the TA3 system has the discretion to determine what nodes successfully match the entry point.

4.2 SAMSON-TA3 Goals and Approaches

Goals. The SAMSON-TA3 system design achieves a number of goals. First, the design addresses a problem of failed queries encountered in earlier evaluations. Because TA1/TA2 linguistic processing is noisy and incomplete (i.e. fails to extract EREs present in the data), a naive query engine running a SIN query will often fail to find any answers. This is often the result of some ERE being mistyped: for example, an event of type Conflict.Attack.FirearmAttack in the text might be extracted with the type Life.Die.DeathCausedByViolentEvents instead. Because TA1 extracts entities, events, and relations, any of these might be mistyped, so a SIN query looking for a particular type might not find it in the data. To ameliorate this problem, SAMSON-TA3 performs relaxed queries, in which the original SIN query is relaxed in order to match a wider range of data.

Second, SAMSON-TA3 is designed to find a variety of distinct, mutually inconsistent hypothesis answers, in keeping with the scoring metric of SM-KBP. Each hypothesis may share edges with another hypothesis, indicating some factual overlap, but each hypothesis must also contain at least one edge not found in the other hypothesis.

Third, because the space of potential hypotheses is large, SAMSON-TA3 is designed to be scalable, capable of traversing millions of distinct hypotheses in less than a minute.

Finally, SAMSON-TA3 is biased to find hypotheses that are connected graphs. While not strictly required, this causes SAMSON to find hypotheses that are more coherent, compact, and informative. For example, SAMSON might discover hypotheses in which multiple victims are slain by the same killer, which is likely to be illuminating for the end user.

Approaches. SAMSON-TA3 implements relaxed hypothesis queries using two different approaches. The first, developed by Raytheon BBN, uses *hierarchical similarity clusters* to find nodes that are similar to the node asked for by a SIN. The second relaxed query approach, developed by partner Polaris Alpha, uses a process of beam-constrained local search to discover nodes similar to those asked for by a SIN. We have substantially completed a Dockerized integration of the two approaches, but for TAC 2019 they operated separately, producing alternate outputs for SINs. We describe each approach in detail next.

4.3 TA3 Related Work

Substantial literature exists on relaxation of queries. Most relevant to our work are recent publications on RDF query relaxation. These works all explore relaxations of RDF queries for the purpose of finding answers that are similar to what was asked for. We are not aware of a work using hierarchical clustering to structure data similarity relationships as we do.

[7] uses clusters to organize query relaxations; these are defined either using ontological relaxation (e.g. replacing Professor by the cluster {Teacher,Researcher} or by substituting node prop-

Table 1: Hierarchical cluster spaces used in SAMSON.

Cluster Space Data Type	Computation Method
String Names	Agglomerative clustering using Lucene
Classes	Analytic
Property Classes	Analytic
Times	Analytic
GeoNames Places	Agglomerative clustering
Entities	Derivative: combine string name and class
Events and Relations	Derivative: combine multiple slot values

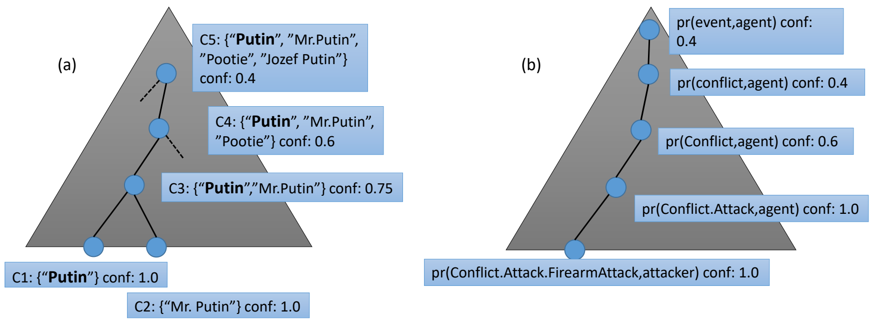

(a)
C5: {"Putin", "Mr.Putin", "Pootie", "Jozef Putin"} conf: 0.4
C4: {"Putin", "Mr.Putin", "Pootie"} conf: 0.6
C3: {"Putin","Mr.Putin"} conf: 0.75
C1: {"Putin"} conf: 1.0
C2: {"Mr. Putin"} conf: 1.0
(b)
pr(event,agent) conf: 0.4
pr(conflict,agent) conf: 0.4
pr(Conflict,agent) conf: 0.6
pr(Conflict.Attack,agent) conf: 1.0
pr(Conflict.Attack.FirearmAttack,attacker) conf: 1.0

Figure 9: Hierarchical cluster spaces for (a) name strings and (b) AIF events. The former is defined using agglomerative clustering; the latter is defined analytically.

erties for nodes and then relaxing the properties (e.g. Directors the same age as Steven Spielberg). [10] relaxes queries by replacing individual nodes with relaxed values, either by ontological relaxation or by node variabilization (substituting a variable for some query node). Like ours, this work scores the relaxation based on similarity; their scoring is based purely on ontological distance. [11] relaxes queries using ontological relaxation, node variabilization, or by breaking join dependencies between variables. Like our work, [8] defines data-specific similarity spaces determined algorithmically and scoring systems.

4.4 BBN Hypothesis Generation

BBN's Hypothesis Generation process, which is implemented entirely in XSB, uses relaxed queries over cluster-level statements.We first organize Unified KB nodes into hierarchical similarity cluster spaces of various types shown in Table 1. Next, we translate Unified KB edges to a cluster-level form, replacing each node in the edge with a cluster from the appropriate cluster space. We similarly translate the SIN to a cluster-level form. In effect, the resulting query relaxes the original SIN by looking not for some node or type mentioned in the SIN, but instead for a cluster containing that node or type. Here, clusters represent sets of items having some similarity to the original item.

4.5 Hierarchical Cluster Spaces

Hierarchical clusters are a well-known method of expressing degrees of similarity between any two items in a given set. We construct clusters using simple agglomerative clustering methods that initially place each item in its own singleton cluster and then repeatedly merge clusters based on inter-cluster distance. Each distance metric is specific to that cluster type. We describe details of each cluster type next.

String Names. We construct a hierarchical cluster space of all string names found in the Unified KB or in the Reference KB. First, we use Lucene text indexing to find scored matches between similar strings. We use Lucene's Damerau-Levenstein similarity with a maximum of 2 swaps as an initial blocking method. Prospective string matches are then evaluated using the more expensive Daro-Winkler metric. For each string we find its 20 most similar strings.

We construct hierarchical clusters by first creating a singleton for each string and then iterating through *all* scored matches in score order. For each match and its associated pair of strings, we merge the two clusters containing those strings. Each new cluster's similarity score (a measure of intra-cluster similarity) is the average of all match scores used in its production. Singleton clusters are given similarity 1.0. This process creates a tree-shaped cluster space in which singletons are at the bottom, the most similar strings are next highest, and larger, lower-similarity clusters are created at each next higher level. For example, as seen in Figure 9(a), the string "Putin" might be a member of multiple different clusters of decreasing similarity going up the tree. [technically, a forest]

Classes. We construct a cluster space of AIF classes analytically, without computing explicit matches or similarity relations between classes. We construct the following event cluster levels and insert AIF event classes in the appropriate clusters:

- top-level category (e.g. event, which contains all events)
- event sub-category (e.g. conflict, which contains all conflict-oriented events)
- three levels of prefix-oriented clusters whose members all share that prefix:
  - (e.g. Conflict)
  - (e.g. Conflict.Attack)
  - (e.g. Conflict.Attack.FirearmAttack)

We construct Entity and Relation class cluster spaces analogously. Cluster intra-similarity scores use a semantic intuition of a probability that two nodes whose classes are in the same cluster are actually the same entity in the world: for example, an intra-similarity score of 0.3 would mean a 30% chance of identicality. These probabilities are currently engineered, but future work could derive them from data.

Property Classes. Property classes are also constructed analytically. LDC Ontology properties mix information about event/relation classes and their slots: for example, Conflict.Attack.FirearmAttack_Attacker refers to the Attacker slot of a Conflict.Attack.FirearmAttack event. We therefore construct a cluster space in which both the class component and the slot component of the property are progressively abstracted. Each property cluster has the form pr(CLASS,SLOT). For each ontology property P having class C and slot S, we make a property cluster for each class cluster CLASS containing C. For singleton property clusters, SLOT $= S$; the value of SLOT for larger clusters are abstracted using engineered categories. For example, in Figure 9(b), we see all the property clusters containing Conflict.Attack.FirearmAttack_Attacker. At the higher levels, the Attacker slot value is abstracted to the more general agent.

Times. Time cluster are constructed analytically to form a cluster space in which the times are less and less similar. For example, singleton clusters contain exact times, while the next level upward captures times whose year, month, day, and hour are all the same, but whose minute can be different. The next level captures times with only the same year, month, and day, and so on. These granularity judgements are a first cut; more subtle temporal divisions (e.g. three-day periods) might be more useful. Since explicitly constructing a cluster for every second in human history is infeasible, we generate only those clusters containing some time mentioned in the Unified KB.

Places. We cluster all places in the Reference KB using their Lat-Long information and simple spatial distance to determine cluster merges. We finished this cluster space, but did not finish code that would reason about it for Phase 1.

Entities. Entity clusters are defined analytically, based on the entity's class and name string. For example, the entity named "Russia" is a member of the singleton cluster en(str("Russia",0), GPE-Country.Country) . We define an entity cluster en(STRCL,CLASSCL) at level k for each STRCL and CLASSCL that are at level k in their respective cluster spaces. We actually explicitly construct only those entity clusters that are non-empty, e.g. the cluster en(str("Putin",0), GPE.Country.Country) has no members, so we do not construct it. This space therefore organizes entities into clusters having less name and type similarity as the cluster grows larger.

Events and Relations. We implemented cluster spaces for Events and Relations based on their types and slot values, e.g. we mode a singleton event cluster whose abbreviated form might look like ev(Conflict.Attack,Person,Person,Tblisi,03/15/04,03/16/04) . For SM-KBP, we did not reason using these clusters because of the difficulty of rigorously handling missing slot values. This problem is analogous to the problem of handling missing data in embedding computation; we plan future work to address it.

4.6 Translating Data and SINs to the Cluster Level

Our approach for exploiting hierarchical clusters in relaxed queries is to translate de-reified edges from section 3.4 to a cluster-level form. Cluster-level translation replaces an edge node (i.e. a predicate or an Entity) by a cluster that node is a member of. Figure 4 shows a sample translation of a de-reified edge to the cluster level.

We create multiple, alternate cluster-level translations of each edge by replacing edge nodes with clusters of varying abstraction. Each KB node is a member of a series of progressively larger clusters. Since the series can be quite long, it is infeasible to, for example, replace the node representing Russia with all the entity clusters it is a member of. Instead, we choose a small number ($k = 4$) of similarity thresholds for each cluster space that are of interest. For each node n and threshold t_k, we then find the (unique) cluster having lowest intrasimilarity that is still higher than t_k. Because cluster intrasimilarity monotonically increases as one goes from a cluster space leaf towards a root, it is straightforward to find this cluster. Using these methods, we therefore obtain a sequence of k clusters containing n that are of increasing size and decreasing similarity.

Armed with these node memberships, we translate de-reified edges. For each edge $e = n1, n2, n3$, we create $k * k * k$ translations of e by systematically replacing each of $n1, n2, n3$ with each of the k clusters found for it in the prior step. Each cluster-level edge can be seen as a relaxation of the original node-level edge; we capture the degree of relaxation by averaging the intra-similarity scores of the clusters used in the edge. The new cluster-level edges are gathered in a Unified Cluster KB that is k^3 as large as the original Unified KB.

For example, in Figure 3.4, we replace the predicate and object nodes from the de-reified edge with clusters (the subject node is an Event, which we do not cluster at present). The predicate node, ldcOnt:Life.Die_Place, is replaced with the property cluster pr(conflict,loc), which is the set of AIF properties specifying the location of some conflict-oriented event. The object node, ta1:entities/a11161.a11554, an Entity, is replaced with the Entity cluster en(str("Georgiev",5),physicalM), which is the set of entities of medium physical size whose name is sufficiently similar to "Georgiev".

For this approach to successfully answer queries, the original SINs must also be translated to cluster level form. This is straightforwardly done by translating the node bindings obtained for SIN entry points to a series of k cluster bindings; the clusters are obtained using the same process used for KB edges.

4.7 Executing the Relaxed Query

Once both the Unified KB data and the SIN are translated to cluster form, we execute the relaxed query in order to generate hypotheses. This query is executed by a meta-interpreter written in XSB. The meta-interpreter runs on top of XSB's built-in interpreter (query engine) and reimplements selected portions of it for additional control. In this case, the meta-interpreter adds multiple special features. Unlike the built-in interpreter, which fails when no match for a SIN edge is found, the meta-interpreter continues looking for matches to the rest of the SIN edges. This feature relaxes the original query by, effectively, allowing parts of it to fail, and returning what matches can be found in the data. This partial matching strategy is necessary to deal with noisy and missing TA1+TA2 data. Additionally, the meta-interpreter performs the translation upon the input SIN described in the prior section. Finally, the meta-interpreter scores every successful answer (hypothesis) and maintains a list of the n best hypothesis answers. We describe the scoring function in the next section. Overall, the meta-interpreter approach allows us to exploit XSB's very powerful query and unification engine, capable of finding millions of hypothesis answers in minutes, while still customizing the portions of the query process described above.

4.8 Scoring and Translation to RDF

The final phase of our pipeline converts the cluster-level hypthesis edges back to node-level edges and then processes these for final output. Conversion of cluster-level edges back to node level begins by replacing entry point clusters with the original entry point node bindings. Next, for each cluster-level edge, XSB looks for an edge from the Unified KB that: (i) is consistent with any existing bindings of SIN variables to nodes; (ii) that contains nodes that are members of the appropriate clusters in the cluster-level edge. In some cases, no edge satisfying (i) can be found, in which case any edge satisfying (ii) is selected (because of the translation and query process, such an edge always exists). Once the node-level edges are obtained, final output processing assembles supporting information from the Unified KB for each hypothesis answer and outputs the hypothesis edges, supporting information, and associated clusters in one RDF file. XSB's built in queries are an efficient method of gathering and massaging this information to meet NIST's requirements. At this stage, to meet NIST's requirements, we generate singleton (co-reference, not hierarchical) clusters for every edge in the hypothesis output. Our processing currently entirely ignores TA2's coreference cluster outputs because we could not coherently integrate them with our hierarchical clustering system.

Our scoring system evaluates each generated hypothesis and optimizes for the four SM-KBP hypothesis objectives of *correctness, relevance, semantic coherence,* and *coverage*. We constrain our approach to be fully correct (i.e. all our returned edges have a TA1 justification). However, we generate varying quality levels for the other three metrics; we optimize for them using the following scoring function, which we explain in detail:

$$S_H = \sum_{e \in H} k_{Seen} * Conf_e / Freq_e$$

To promote overall relevance to the SIN, we promote complete hypotheses by awarding each hypothesis a score for each returned edge e (i.e. each edge matching some SIN edge).

A returned edge may be only partially relevant, in that it matches a relaxed version of the original SIN edge. Furthermore, pairs of returned edges may not be semantically coherent because of the relaxed translation described earlier, which does not always enforce shared bindings. Our scoring metric captures both of these imperfections by incorporating the degree of SIN relaxation $Conf_e$ that was required to find matches in the data. Here, $Conf_e$ ranges between 0.0 (relaxed, less confident, bad) to 1.0 (exact, more confident, good).

$$Conf_e = avg(Conf_s, Conf_p, Conf_o)$$

Confidence is computed simply by averaging the confidence (intra-cluster similarity) values associated with the cluster-level subject, predicate, and object; these are taken directly from the cluster spaces described earlier.

We promote coverage by penalizing hypotheses for using bindings (KB nodes) used by other hypotheses in the k-best set. For each distinct predicate and object node found in any hypothesis in that set, we tally $Freq_p$ [$Freq_o$], its number of occurrences in all hypotheses in the set. At scoring time, we penalize each hypothesis edge by $Freq_e$, which expresses the frequency of the edge predicate and object.

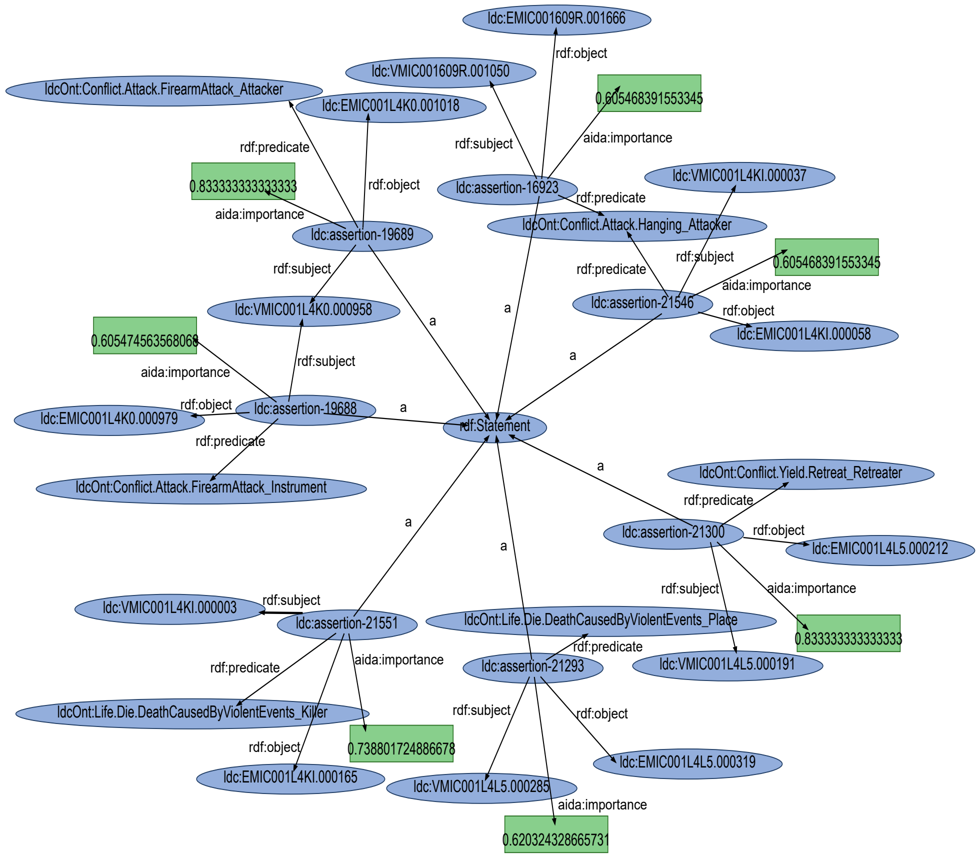


Figure 10: Hypothesis statements in graph form generated by SAMSON using relaxed querying over cluster-level statements. Blue boxes are nodes from the Unified KB. Green boxes are confidence values generated by SAMSON.

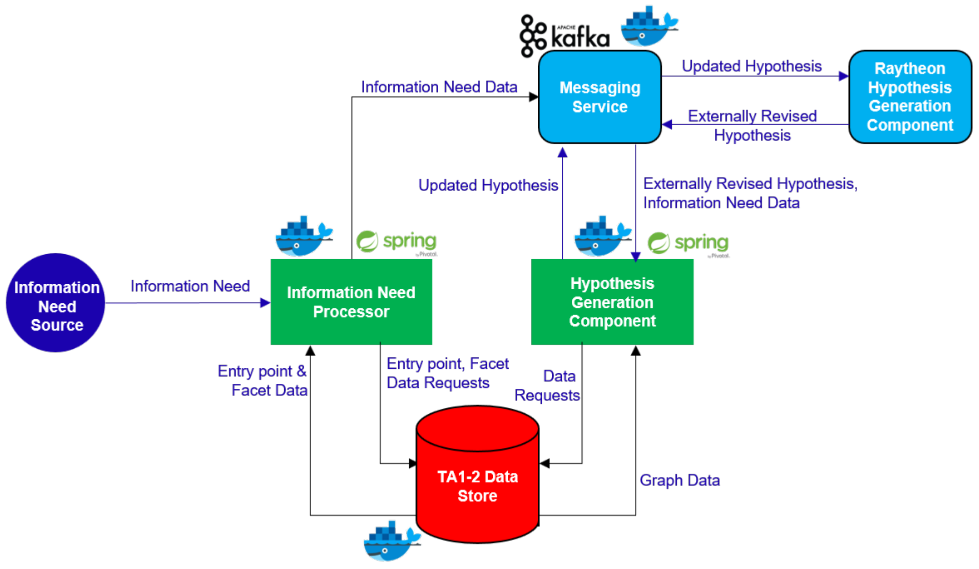

kafka
Messaging Service
Updated Hypothesis
Raytheon Hypothesis Generation Component
Externally Revised Hypothesis
Information Need Data
Updated Hypothesis
Externally Revised Hypothesis, Information Need Data
spring
spring
Information Need Source
Information Need
Information Need Processor
Hypothesis Generation Component
Entry point & Facet Data
Entry point, Facet Data Requests
Data Requests
TA1-2 Data Store
Graph Data

Figure 11: Architecture for Mentieta Miner, Polaris Alpha's Hypothesis Generation engine.

$$Freq_e = avg(Freq_p + Freq_o)$$

Finally, we promote more internally compact, connected hypotheses by offering a score bonus k_{Seen} when s, p, or o for an edge are also seen in another edge from the *same* hypothesis.

4.9 Polaris Hypothesis Generation

Polaris Alpha's Hypothesis Generation approach, named *Mentieta Miner* and shown in Figure 11, has five major components:

- Information Need Source (INS): This simple component receives XML-based INs, processing them to ensure they are well formed. This component can be configured to either receive INs via Kafka messaging or via text file. For the recent Month 18 (M18) test event, IN files were processed.
- Information Need Processor (INP): This Docker component converts INs into internal data structures and queries for relevant entry point and frame edge data. This component is described in more detail below.
- Hypothesis Generation Component (HGC): This Docker component receives processed INs from the Messaging Service and searches for data that can be used to create connected, consistent result graphs that resolve as much of an associated IN as feasible.
- Data Store (DS): Due to the limited scope of the M18 exercise, evaluation data was loaded into a local GraphDB datastore [16]. Test runs were executed using TA1+TA2 datasets from Linguistic Data Consortium (LDC), BBN, and CMU-OPERA. In future exercises, the local datastore will be replaced with a data service provided by the TA2 performer(s).
- Messaging Service (MS): This Kafka messaging service provides support for internal and external communications. Wrapped in a Docker container, Kafka (Apache Software Foundation, 2019) was selected to provide a scalable, fault tolerant, and operational communications infrastructure.

As shown in the figure, Metieta Miner components are wrapped in Docker containers and are configured using Spring. Docker simplifies Metieta Miner distribution, while Spring simplifies algorithm configuration.

4.10 Information Need Processor (INP)

Hypothesis Generation begins with the Information Need Processor. After verifying that a SIN has the proper syntax and structure via the INS, it is passed to the INP for initial processing. The SM-KBP evaluation plan treats multiple SIN frames as separate queries, so the INP first separates frames and operates on them individually.

Each frame has one or more sets of edges with shared subjects or shared objects, as seen in Figure 8, where the edges share a common subject. The INP attempts to resolve as much of a SIN as possible via direct query. After converting to an internal data structure, the INP first attempts to resolve as many entry points as possible. For example, the entry point shown above for ?Sniper1 will be translated by the INP into a SPARQL query designed to retrieve a matching entity from the DS. If a matching entity is found, the INP will extend the entry point data to include any members of an (TA2-generated) AIDA:SameAsCluster to which that entity may belong along with supporting structures. Any member of an entry point entity's SameAsCluster may be used during the next phase of INP processing.

The second phase of INP processing involves attempting to link one or more entry point SameAsCluster members with instances of an event or relation referencing that entry point as part of an IN frame edge. For example, we may find one or more Conflict.Attack.FirearmAttack event instances with a defined FirearmAttack.Attacker relation linked to entities who are members of the resolved ?Sniper entry point SameAsCluster. Like entry point processing, resolved frame edge entity SameAsClusters are retrieved along with any members belonging to those clusters.

Finally, to cast a wider net, semantic softening is applied in which the ontological class structure is used to find semantically similar events to those requested in an IN. Because the AIDA ontology is relatively flat, we've explored adding local extensions that allow us to link, for example, Conflict Attack with Death By Violent Means. Semantic softening is limited to parent and siblings only provided the parent type is not Event or Relation. Semantically softened queries were modeled similarly to those proposed by Lin [13].

4.11 Hypothesis Generation Component (HGC)

Initial hypotheses are formed by taking the cross product of unique entry point SameAsCluster groups with unique resolved frame edge cluster groups. Each hypothesis is then scored. After this initial processing is performed, a search algorithm is used to fully connect the hypothesis graph if needed.

The HGC implements a beam-limited global search algorithm to search for knowledge structures that can be used to connect hypothesis subgraphs together in a logical, well-founded way. Given a hypothesis and an information need, the HGC will attempt to link hypothesis subgraphs together using graph structures queried from the Data Service. A configurable multivariate scoring mechanism is used to create a score (cost) vector for each hypothesis, with the search engine using a greedy approach to select and expand the highest scoring hypotheses until a termination condition has been reached.

The HGC consists of five major subcomponents:

- Graph Search Message Consumer: This subcomponent receives IN data via Kafka and creates local tracking information.
- Graph Search Content Deserializer: This subcomponent builds a search context including the maximum search depth, beam width, branching width, and search termination strategy. The search termination strategy defines the search algorithm's termination condition: search until a maximum recursion depth has been reached, terminate when a solution (connected hypothesis graph) has been found, or a recursion limit has been reached (whichever occurs first). Other search context elements are described subsequently.
- Graph Search Engine: : The search engine uses the search context along with a multivariate scoring function to guide and control the search. In case no hypotheses can be extended into a solution (a connected graph resolving all frame edges), the Search Engine will return the best solutions found so far to through the Messaging Service to the BBN search engine to see if they can be extended to full solutions using the BBN engine.

  In addition to the best solutions list, the Search Engine uses a score-sorted queue to keep track of partial solutions (hypotheses) and their scores. Hypotheses from this queue are extended by the Search Engine to find full solutions. The search context beam width parameter controls at each stage of processing the number of partial solutions pulled from the queue for expansion. For each hypothesis pulled from the queue, the Search Engine extends the hypothesis by finding graph structures from the DS that can be fused to the hypothesis structure. The search context maximum branching parameter is used to control the number of child hypotheses created from a given parent. Each child is scored using a multivariate scoring function and added to the score-sorted queue. Each hypothesis is then scored for confidence, completeness, connectedness, and semantic similarity to the frame edges requested in the IN. Consistency is not yet included in the scoring process but is planned for future releases.

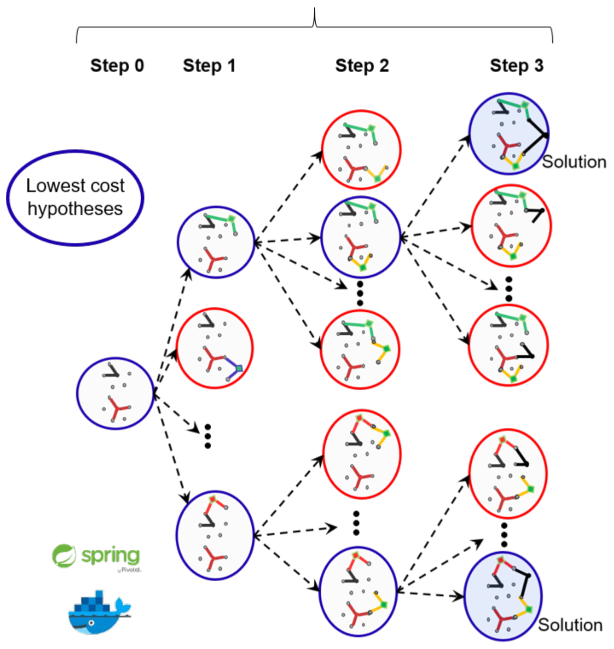


Figure 12: Mentieta Miner search strategy.

  As each child hypothesis is created and scored, an extraction function is used to check if the resulting child is a solution to the IN. If so, it is passed to the Search Engine Serializer for transmission to the BBN component. Note that new scoring functions and associated weights can easily be added through Spring integration, and the size of the queue can be controlled via a search strategy parameter. The original IN is available to the Search Engine as needed. Search engine operation is graphically depicted in Figure 12.
- Search Engine Serializer: This subcomponent translates full and partial solutions from internal representations into an internal JSON representation and serializes the data for transport. Note that the internal representation is relatively spartan; structures required by DARPA and NIST such as type assertions are added via an external rehydration process.
- Search Engine Producer: This subcomponent uses Kafka to send hypothesis / solution messages and status messages.

4.12 Future Work

We plan several areas of future work. First, we will complete integration of BBN's and Polaris Alpha's respective TA3 approaches, which we hope will complement each other. BBN's approach emphasizes rigorous relaxation of SIN nodes and produces smaller hypotheses, while Polaris Alpha's approach emphasizes more opportunistic proximity-based relaxation and produces more hypothesis content. We will also optimize each approach and anticipate greatly increased accuracy and efficiency.

For BBN's engine, we will optimize our query meta-interpreter to lazily operate over the original node-level statements, querying node cluster memberships on demand. This optimization would obviate the need for creating the (n^k) larger cluster-level statements. We will also perform better query optimization: the interpreter currently attempts to match SIN entry points and edges in an arbitrary order, leading to unnecessary backtracking. Our enhanced engine will match against entry points and associated edges in parallel, constraining hypothesis growth to avoid repeated backtracking. Relatedly, we will introduce beam search logic to its query process so that it abandons partial query answers unlikely to produce a valuable hypothesis.

BBN will also explore more useful cluster spaces. Our current system's performance is degraded by the relatively unsophisticated Lucene text matching, we will shift to using BERT and other textual embedding and matching techniques. We also plan to use Inductive Logic Programming (ILP) techniques to learn cluster space definitions for complex ERE data types. These spaces would cluster ERE instances using learned logical rules built from a vocabulary based on the cluster definitions described in the current paper. Our hope is that this system would learn appropriate levels of abstraction and produce higher quality hypotheses. This approach would also incorporate TA2 clusters as vocabulary items, enabling us to rigorously assimilate TA2 outputs instead of ignoring them as we do now.

For Polaris Alpha's engine, we will explore query optimization strategies analogous to those described above. We will also explore horizontal scaling of the core engine. Finally, we will modify the semantic softening done in the INP to make use of Raytheon/BBN's extracted semantic relationship data. Doing so should not only help address limitations associated with the relatively flat AIDA ontology, but also help resolve previously unresolved entry points.

5 Conclusion

Our SAMSON team developed novel solutions to the problems of KB Construction and Hypothesis Generation using noisy, incomplete semantic data. Our SAMSON system computes rigorous solutions over datasets of over 100M triples in only a few hours for TA2 and less than an hour for TA3. Both our TA2 and TA3 methods are novel, early-stage, and would greatly benefit from further experimentation and optimization, likely making them more competitive in future evaluations.

References

[1] M. Ankerst et al. Optics: ordering points to identify the clustering structure. In *ACM Sigmod record*, volume 28 of number 2, pages 49–60. ACM, 1999.

[2] S. Auer et al. Dbpedia: a nucleus for a web of open data. In *The semantic web*, pages 722–735. Springer, 2007.

[3] P. Bojanowski et al. Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics*, 5:135–146, 2017. ISSN: 2307-387X.

[4] A. Conneau et al. Word translation without parallel data. *arXiv preprint arXiv:1710.04087*, 2017.

[5] J. Devlin et al. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805, 2018. arXiv: `1810.04805`.

[6] M. Ester et al. Density-based spatial clustering of applications with noise. In *Int. Conf. Knowledge Discovery and Data Mining*, volume 240, page 6, 1996.

[7] S. Ferré. Answers partitioning and lazy joins for efficient query relaxation and application to similarity search. In *European Semantic Web Conference*, pages 209–224. Springer, 2018.

[8] A. Hogan et al. Towards fuzzy query-relaxation for rdf. In *Extended Semantic Web Conference*, pages 687–702. Springer, 2012.

[9] M. Honnibal and M. Johnson. An improved non-monotonic transition system for dependency parsing. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1373–1378, 2015.

[10] H. Huang et al. Computing relaxed answers on rdf databases. In *International Conference on Web Information Systems Engineering*, pages 163–175. Springer, 2008.

[11] C. A. Hurtado et al. Query relaxation in rdf. *Journal on Data Semantics X*:31, 2008.

[12] J. Johnson et al. Billion-scale similarity search with gpus. *arXiv preprint arXiv:1702.08734*, 2017.

[13] D. Lin. Automatic retrieval and clustering of similar words. In *COLING 1998 Volume 2:*

The 17th International Conference on Computational Linguistics, 1998.

[14] W. McKinney. Data structures for statistical computing in python. In *Proceedings of the 9th Python in Science Conference*, pages 51–56, 2010.

[15] T. Oliphant. NumPy: a guide to NumPy. USA: Trelgol Publishing, 2006–. URL: `http://www.numpy.org/`. [Online; accessed ¡today¿].

[16] ontotext.com, 2019. URL: `http://graphdb.ontotext.com/`.

[17] A. Paszke et al. Automatic differentiation in PyTorch. In *NIPS Autodiff Workshop*, 2017.

[18] F. Pedregosa et al. Scikit-learn: machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[19] B. Perozzi et al. Deepwalk: online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 701–710. ACM, 2014.

[20] M. E. Peters et al. Deep contextualized word representations. *arXiv preprint arXiv:1802.05365*, 2018.

[21] E. Prud'hommeaux and A. Seaborne. SPARQL Query Language for RDF. W3C Recommendation, 2008. URL: `http://www.w3.org/TR/rdf-sparql-query/`.

[22] P. Ristoski et al. Rdf2vec: rdf graph embeddings and their applications. *Semantic Web*, 10(4):721–752, 2019.

[23] A. Rosenberg and J. Hirschberg. V-measure: a conditional entropy-based external cluster evaluation measure. In *Proceedings of the 2007 joint conference on empirical methods in natural language processing and computational natural language learning (EMNLP-CoNLL)*, pages 410–420, 2007.

[24] T. Swift and D. S. Warren. Xsb: extending prolog with tabled logic programming. *Theory and Practice of Logic Programming*, 12(1-2):157–187, 2012.

[25] M. Wick. *GeoNames*. GeoNames, 2006.

[26] Z. Yang et al. Xlnet: generalized autoregressive pretraining for language understanding. *CoRR*, abs/1906.08237, 2019. arXiv: `1906.08237`. URL: `http://arxiv.org/abs/1906.08237`.

[27] D. Zhang et al. Attributed network embedding via subspace discovery. *Data Mining and Knowledge Discovery*:1–28, 2019.