

Similarity in Semantic Graphs: Combining Structural, Literal, and Ontology-based Measures

Lindsey Vanderlyn , Carl Andersen, Plamen Petrov

Raytheon BBN Technologies, Arlington, VA, USA
 {lvanderl, canderse, ppetrov}@bbn.com

Abstract—Semantic graphs provide a valuable way to represent data while preserving real world meaning. As these graphs become more popular for storing large quantities of data, it is important to have methods of determining similarity between nodes in the graph. This paper extends previous structural similarity algorithms by taking advantage of meaning contained in a graph’s literals and the graph’s ontology and allowing users to control how much each type of similarity effects overall scores. Preliminary tests indicate that including these sources of similarity increases scores in way that is better aligned with human intuition.

Keywords: Semantic Graphs, Graph Similarity, Semantic Similarity, Structural Similarity, Ontological Similarity, Literal Similarity, Intelligence Problem Decomposition, Random Walks

I. INTRODUCTION

One of the challenges faced with increasingly large data sets is analyzing the information. In particular, finding similar pieces of data from within a larger data set can prove difficult. This problem is especially present when analyzing semantic graphs, which preserve the meaning and context of data by representing objects in terms of their relationships. Raytheon BBN Technologies (BBN) has been expanding previous research to fully utilize information encoded in a semantic graph.

The present paper proposes SSDM+, an algorithm which extends existing techniques for computing the *structural similarity* between two graph nodes. Structural similarity techniques measure node pair similarity by examining the similarity of the pair’s respective subgraphs. In the following sections, we will give a brief introduction to semantic graphs, detail the motivation behind this research, describe the prior work we build on, explain the new algorithm we have developed, and demonstrate the application of our algorithm to the decomposition of a notional intelligence problem of “Illegal Fishing”.

II. SEMANTIC GRAPHS

Resource Description Framework (RDF) graphs are comprised of two types of elements: resources and literals. Resources are things which can be described, such as objects and relationships. To remove ambiguity about which object a statement is being made, resources are given globally unique resource identifiers. Literals, in comparison, are used to annotate resources with data values such as strings and integers. There is no expectation that literals will be unique, so with RDF statements in the form of subject-predicate-object triples, literals are never allowed to be the subject or the predicate.

Another key feature of RDF graphs is the inclusion of ontology data in the graph itself. Ontologies are often compared to relational database schemas. Ontologies define classes, relationships, and their inheritances. Through these definitions, all of the statements defined for a class are added to the graph (inferred) every time a user creates a new instance of that class. For example, when given an ontology, a reasoning engine will infer that instances of a class are also instances of all of the class’s ancestors.

III. MOTIVATION FOR THIS WORK

This work is primarily motivated as part of an effort to develop a tool for creating intelligence problem decompositions represented as semantic graphs. In this context, finding similarity between nodes in a semantic graph could allow analysts to collaborate and reuse portions of existing problem decompositions when working on a new problem.

A. Semantic problem decomposition

In order for analysts to determine what information needs to be collected for a particular intelligence problem, they must first break the problem down in a logical and systematic way - a process we refer to as problem decomposition. To illustrate this, we will use

a fictitious intelligence problem of “Illegal Fishing in Hawaii” shown on Fig. 1. The first step is to determine the **states** in which entities related to the problem (e.g., fishing boats) can exist. For example, a fishing boat might be either “in port” or “in a legal fishing zone”. Next, the analyst must specify how to tell if an entity is in a particular state by creating **indicators**. Indicators must be boolean statements such as “boat moving” or “outriggers deployed”. The analyst then specifies the observable phenomena, or **stimuli**, that needs to be collected in order to determine if an indicator is present. Next the analyst defines sensor system specific collection parameters, called **observation needs** such as resolution, location, and time to collect the stimuli. Finally, an analyst can specify **algorithms** which combine input from one or more observation needs to determine whether an indicator is present.

Semantic graphs are a good way to represent problem decompositions because they provide an intuitive way for analysts to capture the relationships from a mental model of the problem into a format, which is both machine-readable and human-understandable. In addition, storing the decomposition as a graph preserves the semantics of the relationships between decomposition components. Finally, these graphs can also be reasoned over to determine completeness, decomposition alternatives, or efficiency of collection.

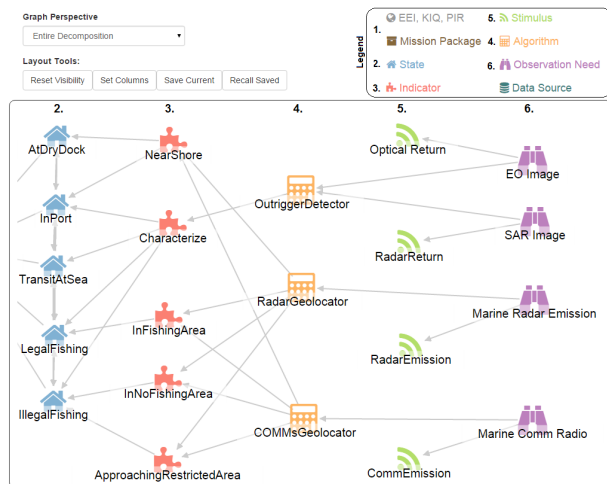


Fig. 1: Example of a simple semantic problem decomposition graph for the intelligence problem “Illegal Fishing in Hawaii”. Nodes in column 1 represent intelligence problems (not shown), column 2 - States, column 3 - Indicators, column 4 - Algorithms, column 5 - Stimuli, and nodes in column 6 represent Observation Needs. Edges represent logical relationships such as “Supports” or “State Transition”.

IV. PREVIOUS ALGORITHMS

Our work on SSDM+ derives from three earlier works. These are SimRank [2], the work of Fogaras and Racz

in their paper: Scaling Link-Based Similarity Search [5], and the Semantic Similarity Distance Metric (SSDM) [7]. These algorithms share the following desirable qualities:

- They are domain-independent, meaning that they can be applied to any data representing relationships between entities.
- They can be computed efficiently over very large datasets, in contrast to algorithms which scale very poorly, such as ones which rely on Singular Value Decomposition [4].
- They have the ability not only to determine the similarity of any two given nodes, but also to generate a list of entities which are most similar to one specified by a user.
- The computations are easily understood, meaning that similarity scores generated can be explained based on the actual computation performed - something which can be difficult with more abstract calculations.
- The algorithms look beyond a node’s immediate neighbor to create a broader knowledge of the entity’s structural context.

We provide a more in depth explanation of each of the previous algorithms below followed by an overview of the changes we have made with SSDM+.

A. SimRank

The key insight of SimRank is that two nodes are similar if they are connected to similar nodes [2]. This simple idea can be translated mathematically as: the similarity score between two entities is the average pairwise similarity of their neighbors, scaled by a decay factor.

A concrete example of this can be seen with movie data. Consider the example in Fig. 2.

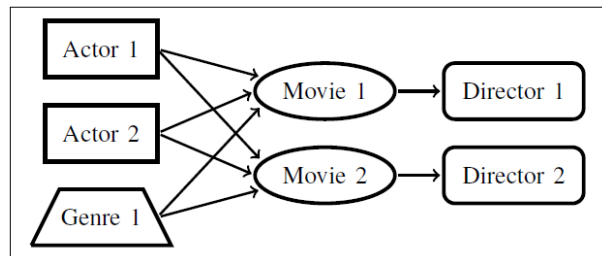


Fig. 2: Example of a semantic graph [7] with information on movies. Director 1 and Director 2 are similar because they directed similar movies, although the directors themselves are not connected

Movie 1 and Movie 2 appear to be similar to each other because they are in the same genre and they both have two of the same actors. Furthermore, although

Director 1 and Director 2 do not share any nodes, they can also be considered similar because they directed similar movies. SimRank provides a formalization of this type of idea.

Each pair’s similarity is dependent on many other pairs because of the recursive definition of SimRank. On small datasets, the system can be solved with an iterative algorithm. On large datasets, it can be solved using an efficient approximate method outlined by Fogaras and Racz in [5]. The SSDM calculation and our extensions are based on this efficient approximate method.

B. Fogaras and Racz’s extensions

The algorithm outlined by Fogaras and Racz relies on the mathematical notion of a *random walk* through a graph, in which an abstract walker steps from node to node through the graph by following random edges [5]. In the original SimRank paper, Jeh and Widom [2] observed that the SimRank score of two nodes can be approximated from the expected meeting time of two random walkers starting at those two nodes; a longer expected meeting time corresponds with a lower SimRank score. Fogaras and Racz used this observation to develop an efficient and scalable algorithm for calculating similarity scores. In their algorithm, one walker is initialized per node and at every time step, each walker steps across one edge. To reduce the number of computations required, walkers converge (are treated as a single walker) when they meet at the same node. Fogaras and Racz found that this type of convergence does not reduce the correctness of the approximate calculation.

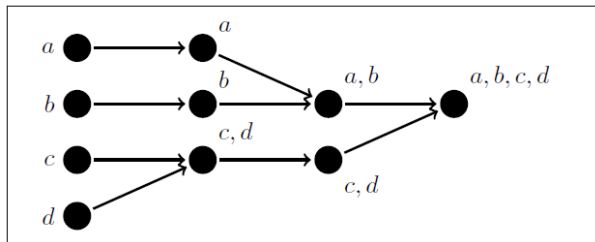


Fig. 3: Diagram of walker convergence [7]. Walkers *a*, *b*, *c*, and *d* begin as independent. As the progress through the graph (left to right) they meet each other and converge.

Additionally, Fogaras and Racz’s algorithm incentivises walkers to converge if they are near each other. Convergence is encouraged as a result of how a walker chooses its next step. In this model, all of the edges in the graph are randomly assigned an index and walkers will always choose the edge with the lowest index to step to. This shuffling of indexes is universal. This means that if walkers share multiple of the same possible next steps, and the walkers select a shared node as the next step, they must choose the same shared node. Restated: the

set of shared possible next steps for the two nodes only contains one edge with the lowest index. Therefore, if each of the walkers chooses the edge with the lowest index from this set, they will choose the same edge and meet.

The output of one run of Fogaras and Racz’s scalable SimRank algorithm is a collection of *fingerprint graphs* which store the convergence data for each of the random walkers. Each graph stores the node a walker began on, the node it merged into, and how many steps it took for that convergence to occur. These fingerprint graphs are precomputed and are collected to be stored in database which can be queried at runtime. Separating the fingerprint graph generation and querying allows real time similarity queries to be quite rapid.

C. SSDM

SSDM [7], also developed by BBN, was closely related to SimRank with two principal differences. The first difference is that SSDM seeks to be independent of ontological choice, meaning that the directionality of each edge (predicate) is ignored.

The second involves the semantics of RDF graphs. Whereas SimRank is a measure over unlabeled directed graphs, SSDM incorporates the semantic labels given to edges. This makes it well-suited to semantic graph data, which is organized in subject-predicate-object triples. Subjects and objects are equivalent to nodes in the graph and predicates are equivalent to edges. In RDF graphs, these predicates, or labeled edges, describe how the nodes they connect are related to each other. In order to utilize the meaning contained in these edge labels, the SSDM algorithm only considers two walkers to have met if they arrived at the same node having traveled through an identical list of edges to get there. This adds a stricter definition of similarity; it is no longer good enough that two nodes were connected to the same third one, they now had to be connected in the same (semantic) way.

V. SSDM+: INCORPORATING LITERAL AND ONTOLOGY-BASED SIMILARITY

Our new algorithm builds off of previous work with SSDM by allowing literals to influence similarity scores, relaxing the need for walkers to meet at the exact same resource in order to converge, and creating a stricter approach to how the ontology influences similarity. This is motivated by the desire to create a cohesive model for including literal and ontological similarity in a larger structural similarity framework. Additionally, we removed the weighting of similarity scores based on the time it takes walkers to meet (decay), because the identical paths restriction already significantly decreases the probability of any walkers meeting after very few

steps and we found that further decay was not necessary. Otherwise we retained the same functionality and methods as the original SSDM algorithm.

In SSDM, similarity was calculated only taking into account the resources in the graph; literals in the graph were completely ignored. Additionally because of the strict definition of convergence, nodes which humans might consider to be very nearly the same (such as multiple instances of the same class) could not contribute directly to the similarity scores. This approach does not fully utilize the ontological information stored in a semantic graph. String literals often serve as meaningful labels for nearby nodes; numerical literals store quantities that tend to be of similar magnitude for nearby nodes. Therefore ignoring literals causes some nodes to appear much more similar than a human might consider them and the requirement for walkers to walk the exact same predicate path, prevented inferred inheritance from sufficiently representing the ontological similarity of two nodes. In particular, because of the number of inferred classes a resource might have, it was highly probably that two nodes, which shared an ancestor, might walk a path to that ancestor, but each take a different number of steps to get there. Additionally, many of the inferred classes are very high level concepts (e.g. owl:Thing) and the number of instances of these classes is high enough in a graph that even if two walkers did converge at one of these parent classes, a human would not consider that convergence to have added any similarity.

With SSDM+, we combine both the literal and ontological similarity into a larger structural approach. There may be other existing works which make use of literal and ontological similarity, but we are unaware of any others which do so as part of a cohesive structural similarity measurement.

A. Literal Similarity

In order to incorporate literal similarity, we had to loosen the definitions of convergence present in the original paper [7]. Rather than only converging when two walkers had reached the same resource (referred to here as "physical" convergence), walkers could converge if they reached two literals that were sufficiently similar. In this case, we defined similarity for numeric literals as the ratio of the smaller over the larger - or percent. The similarity for all other literals - which for simplicity were converted to strings - was calculated as a Levenshtein distance between the two, normalized for their sizes. Levenshtein distance is a measure of structural similarity between two strings by measuring the number of substitutions, insertions, and deletions required to turn one string into the other. These metrics work well with the data sets we tested, but for graphs which contain

long strings or vastly different types of data, alternative to Levenshtein distance (e.g., [6]) can be utilized.

B. Ontological Similarity

In an analogous way, we added ontology-based similarity by allowing walkers to converge if they stepped onto resources which we considered to be ontologically similar. In this case, we define "ontologically similar" as: sharing a most-distant-salient ancestor, with salience defined as:

$$Salience = 1 - (instances/total) \quad (1)$$

where *instances* is the number of instances of a class and *total* represents the total number of nodes in the graph. This number represents how unique each class is, or how much of the graph is **not** made up of instances of that class.

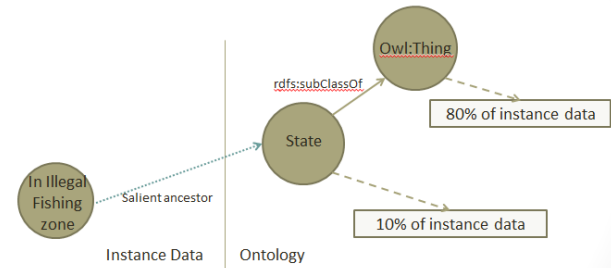


Fig. 4: Example of finding the most-distantly-salient ancestor. The node "In Illegal Fishing zone" is an instance of a State (fairly unique). State is also a subclass of owl:Thing, but because that is too common to be considered salient, "In Illegal Fishing zone's" most-distant-salient node is State.

To find the most-distant-salient ancestor, we precomputed the salience of each class and traced its ancestry until we could find the most distantly related ancestor which remained more salient than a user defined cutoff (see example in Fig. 4). Defining similarity this way means that all instances of a class that is considered to be salient (or instances of that class's children) will have the same most distantly related ancestor, thus reducing the number of comparisons needed to find similar nodes. Additionally, when we implemented the ontology-based matching, we removed nodes representing inferred types from the graph, which allows the user to fully control the effect of ontology-based matches on the final similarity score and removes the possibility for nodes to match based on a shared, unsalient class.

C. Five Types of Convergence

To make this tool as flexible as possible, we compute an overall similarity score composed of five different similarity scoring methods. The user assigns weights

for each type, allowing them to emphasize the method they believe will deliver the most accurate results in the current context.

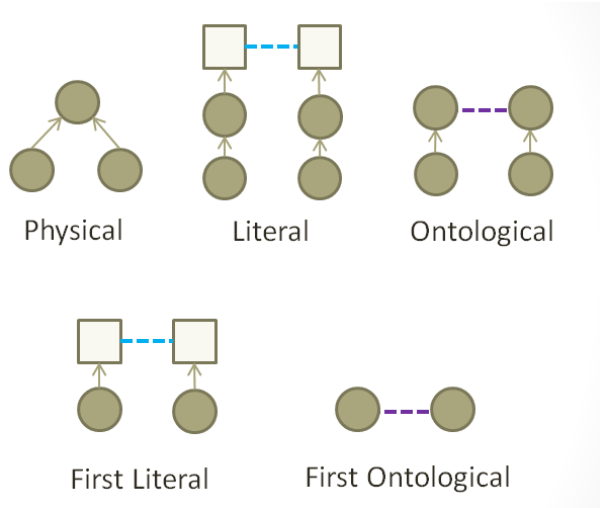


Fig. 5: Visualization of different convergence types. Arrows represent edges in the graph connecting nodes. Light (blue) dotted lines indicate literal similarity of unconnected nodes. Dark (purple) dotted lines indicate ontological similarity of two unconnected nodes.

Walkers are considered to have converged when:

- 1) *Physical convergence*: two or more walkers meet at the same resource
- 2) *Ontology-based convergence*: two or more walkers step onto nodes whose classes share a most-distant-salient ancestor
- 3) *Literal based convergence*: two or more walkers step onto literals that are more similar than a user defined cutoff
- 4) *First ontology-based convergence*: the nodes they start on are considered to be ontologically the same - because the graph sizes can become quite large, this is only evaluated after the first step
- 5) *First literal based convergence*: walkers step onto similar literals after the first step - this is considered significant because it is the only step where the literals are guaranteed to be connected to the node a walker started on

The last two types of convergence are separate from the others because they are the only two types of convergence that happen based on the original nodes a walker started on. If a user were to query the similarity between two nodes, while they are important to finding the overall structural similarity, those two are the only types of similarity that are uniquely about the queried nodes.

D. Integration of new types of convergence

As with the original SSDM, finding similarity is broken into two stages: a preprocessing stage where fingerprint graphs (trees) are generated from the convergences that occur during random walkers and a runtime stage where these trees are queried. The changes we have made affect the former. SSDM+, our new algorithm, maintains the random walk paradigm of the original SSDM algorithm and continues to use convergences to generate trees. The primary difference is the way walkers can converge. By extension, trees now also contain an additional piece of information: the type of convergence which has occurred. Tracking the convergence type allows a user to control how much each type of convergence affects the similarity scores they receive.

By preserving the random walks paradigm, we gain a consistent framework for assessing all five types of similarity. From a more practical view, however, the requirement that walkers must travel the same predicate path greatly reduces the number of comparisons which must be completed to find literal and ontological similarity.

In addition, the order which we check convergences was also influenced by the need to reduce the number of comparisons, a need which becomes increasingly important as datasets grow. We first check the "physical" convergences because they are the most common type of convergence. We then check for literal convergences and then for ontological convergences. This order matters less because we separate the literals and resources so the two types of convergence are independent of one another. In order to reduce the number of comparisons we must perform, we changed the way "physical" convergence is calculated. In the new method, we sort all walkers by the path they have traveled, then split within those groups, by the current location of the walker. With the nodes that did not converge, but did walk the same path, we split the walkers based on whether they are located at a resource or a literal and look to see if they can converge via ontological similarity or literal similarity. This is a more efficient method (as compared to the original SSDM), when literal and ontological similarity must also be determined. The main drawback to this order of comparisons is that literal and ontological similarity are not calculated until after the first step has been taken. This greatly increases the speed of the calculation, but does ignore ontological and literal similarity between two original nodes if their walkers did not take the same first step. Given the high number of random walks we perform for each graph, we consider this acceptable.

VI. TESTING AND TUNING

In the course of the development of the SSDM+ algorithm, we executed multiple test cases comparing

the results of the algorithm against human intuition on which nodes should be considered to be similar. Based on these empirical results, we developed a procedure for tuning the similarity parameters, as outlined below.

A. Determining literal and ontological cutoff similarities

To determine some baseline values for similarity thresholds, we experimented with both the literal and ontological cutoffs. Cutoffs represented the minimum similarity needed for a user to consider two literals or two resources to be the same for the purposes of allowing convergence. We recorded the time it took to generate trees and the similarity scores between three sets of nodes. These experiments allowed us to optimize the shortest time needed to generate the trees for the highest increase in similarity from the added literal and ontological matching. For all of the tests, the weighting on each type of convergence was kept equal so that only the cutoffs were being varied. From our experiments, we found that literals needed to be at least 60% similar and classes could be considered salient if less than 60% of the instance data was an instance of that class. With higher cutoffs, the time needed to generate the trees began to increase dramatically. With lower cutoffs, similarity scores did not match very well with human predications. It should be noted that these baseline values were generated using mock intelligence problem decompositions and may need to be adjusted for other types of datasets.

B. Determining weights for physical, literal, and ontological convergences

In order to see the effect of allowing literal and ontological matching, we created a series of decompositions. We then tested them to see how changing the weighting of each type of similarity affected which nodes were returned as similar when queried and how similar the algorithm considered them. As a first test, we constructed ten identical graphs whose only difference was in the literal labels associated with each node. The goal of this test was to show that when a node was selected, the most similar resources would be the corresponding nodes in the other graphs.

Once we verified that this was true, we started to test on graphs which were generated from the same template, but were not identical. Templates in the Semantic Problem Decomposition (SPD) tool are fragments of a decomposition that can be reused. When they are copied into a new graph, they retain all of their properties, except that all of the nodes in the graph are given new randomly generated Universal Resource Identifiers (URIs), which are globally unique. These graphs were the primary data we used to test the weighting. In these

tests, we used a smaller dataset - four graphs, rather than ten - and aimed to find parameters which returned results that we, as humans, understood to be similar. In order to get more universal results, we made sure to test several different types of nodes and observe the results. This proved to be interesting as some of the nodes were more influenced by literals - in both positive and negative manners - than others. From these tests we were able to find the weights which had the greatest improvement over the results generated from only considering physical convergence.

Results of this type of tuning may be dependent to the datasets we used. We believe that, in order to use this for other datasets, it would be important to run a similar test to choose appropriate weights.

VII. APPLICATION: ILLEGAL FISHING PROBLEM DECOMPOSITION

A tool for finding similarity between nodes in a semantic graph can be useful in a variety of settings. In this case, however, we are primarily interested in how it can aid in rapid development of intelligence problem decompositions via the retrieval and reuse of existing decomposition structures. While representing problem decompositions in a semantic way has many benefits, it can also be a time consuming process to manually input all of the relevant nodes. We envision using SSDM+ to suggest nodes from existing problem decompositions which are similar to a node selected by a user. In this way, the user would be able to look at the graphs associated with each suggestion and, if desired, copy the suggested node and its children into the new decomposition.

This interaction reduces the amount of time it takes an analyst to perform their job, but also gives them more context for generating a new decomposition. In addition to saving the analyst time, a tool like this also provides analysts the opportunity to see how their peers solved similar problems. This type of knowledge sharing could allow analysts to envision new ways of completing their decomposition tasks. This latter feature has the potential to be especially significant, given the limited sensor resources for collecting stimuli.

In this section we use the mock decomposition problem of illegal fishing which we described earlier in the paper and demonstrate some of the preliminary results we have been generating.

A. Results

The results shown in this section were determined by comparative analysis of similarity suggestions. We created four decompositions from the same starting template and modified them by adding and deleting

Top Three Suggestions of Most Similar Nodes and Their Similarity Scores for the Node: “In Illegal Fishing Zone 4”

Types of convergence	First suggestion (score)	Second suggestion (score)	Third suggestion (score)
Physical (SSDM)	In Illegal Fishing Zone (0.57)	In Illegal Fishing Zone2 (0.46)	In transit3 (0.33)
Physical and Literal	In Illegal Fishing Zone (0.57)	In Illegal Fishing Zone2 (0.46)	In transit3 (0.33)
Physical and Ontological	In Illegal Fishing Zone (0.85)	In Illegal Fishing Zone2 (0.75)	In Illegal Fishing Zone3 (0.73)
Physical, Literal, and Ontological	In Illegal Fishing Zone (0.85)	In Illegal Fishing Zone2 (0.75)	In Illegal Fishing Zone3 (0.73)

TABLE I: Top three results from similar decompositions when querying for most similar nodes to the “In Illegal Fishing Zone” node. The far left column describes the types of convergence which contributed to the scores and the remaining columns represent the labels of the highest scoring nodes and their similarity scores. All of the suggestions are the same class (States), but SSDM+’s inclusion of Literal and Ontological convergence produces scores which align better with human understanding, and does find the “In Illegal Fishing Zone” node from each graph to be the top three most similar.

nodes and connections. This gave us four similar, but not identical decompositions. We then generated suggestions for several nodes varying which types of convergences contributed to the similarity scores. A sample of the results can be seen in Table I and Table II where we show the top three suggested nodes and their similarity scores for two of our test nodes. For simplicity, if a type of convergence was included the weight was set to 1 and if it was not included, the weight was set to 0.

In the first example, both algorithms perform similarly. The main difference is the magnitude of the scores. It should be noted, however, that while both algorithms produced results which were all of the same class (State) as the node from which the query was generated, the modified algorithm - when ontological similarity was included - was able to provide a “In Illegal Fishing Zone” node for each of the top three results. The second example, though shows a more extreme distance. The results of the modified algorithm, are from the same class (Indicators) as the node from which the query was generated, while the results from the original algorithm are Stimuli - a class which connects to Indicators.

From our experimentation, we found that are two main improvements offered by SSDM+. The first improvement is that the top suggestions are more frequently closer matches for the node which has been queried. The second improvement stems from the addition of terms for new types of similarity which were previously ignored, resulting in higher scores that align more closely with how a human might perceive the similarity. For example

Top Three Suggestions of Most Similar Nodes and Their Similarity Scores for the Node: “Outriggers not deployed 4”

Types of convergence	First suggestion (score)	Second suggestion (score)	Third suggestion (score)
Physical (SSDM)	Comms2 (0.40)	SAR (0.38)	SAR4 (0.37)
Physical and Literal	Comms2 (0.40)	SAR (0.38)	SAR4 (0.37)
Physical and Ontological	Outriggers not deployed (0.69)	Outriggers deployed2 (0.51)	Transponder On3 (0.42)
Physical, Literal, and Ontological	Outriggers not deployed (0.70)	Outriggers deployed2 (0.51)	Boat Moving2 (0.42)

TABLE II: Top three results from similar decompositions when querying for most similar nodes to the “Outriggers not deployed” node. The far left column describes the types of convergence which contributed to the scores and the remaining columns represent the labels of the highest scoring nodes and their similarity scores. SSDM+’s use of Literal and Ontological convergence performs in a more human understandable way here: suggesting nodes which are all of the same class (Indicator) as the node for which the query was generated, in contrast to the original SSDM algorithm which suggests only Stimuli nodes.

in Table I, the top suggestions are all for the corresponding nodes from other similar decomposition models. These top-suggestion nodes have nearly the same labels as the queried node and, in this test, have identical connections. The similarity scores increase dramatically with ontological similarity turned on, demonstrating the value-added of ontological matching; however, even when considering non-ontological matching the the top three suggestions of SSDM+ are still as good or better than that of SSDM.

While more extensive testing would provide a more definitive picture, our preliminary results indicate that supplementing physical convergence with literal and ontological convergence does increase the accuracy of similarity calculations and the relevance of the results generated. Additionally, we believe that the incentivized Physical convergence described in Section V may unequally weight the Physical component of the score. We plan to investigate and address this issue in future work.

VIII. FUTURE WORK

There are several possible directions for future work. The first, and most pressing, is a more rigorous comparison between SSDM+ and the original SSDM algorithm. A major difficulty in performing such a comparison is the lack of concrete ground truth similarity values to compare our algorithm results to. To that end, there is a need to develop a repeatable system for quantifying which nodes humans consider to be similar. However, due to the subjective nature of such measurements,

constructing ground truth data based on user judgments presents many challenges. An alternative approach could be to compare the similarity results from our SSDM+ algorithm to similarity derived from entity attribute comparison - comparisons of the presence/absence of certain attributes. If we choose to pursue this path, there are several well-established algorithms [8] that we could use. However, since the data we have been using so far consists of only a handful of main classes, such comparisons may not be as useful.

The second area for improvement is in tree generation. One of the goals prompting this research was to develop a way for intelligence analysts to reuse portions of existing decompositions when developing new mission packages. To aid in that process, we hoped that analysts could click on a node they had added to their graph and query for all similar nodes among all of the mission packages that had already been developed. This is not yet possible because of the latency involved in the tree (re-)generation stage. In order to re-run the entire graph analysis, it takes more than a minute on small graphs with size of approximately three hundred nodes. Such a long delay is unlikely to be tolerated by an analyst in interactive mode. We are currently investigating ways to reduce latency, including further optimization of the algorithm and applying the algorithm incrementally as nodes are added to a graph.

We also plan to explore more accurate techniques of determining literal similarity. In particular, at present, we analyze only the structure of textual literals rather than looking at the meaning of that text. Finally, although we have relaxed the strictness in matching nodes, we still require that walkers walk an identical predicate path. It may be, that in a similar way to the additional insight gained by allowing ontological matching, we could also gain a better measure of similarity by relaxing the predicate restrictions so that walkers would also be allowed to travel similar rather than identical predicate paths. The challenge with this would be to find a computationally efficient way of relaxing the restrictions because performing something similar to our ontological check for convergence would be too time intensive for a dataset of any reasonable size.

Additionally, the role of literals is currently undermined by the fact that we encourage convergence in the same way as the original SSDM algorithm which biases physical convergences to occur. In doing this, literal matches occur much less frequently and thus have less of an effect on the total results. Due to the low repeatability, however, we found it was not feasible to stop encouraging convergence. To fully utilize similarity from literals in the graph, an alternative to the way convergence is encouraged would need to be developed.

IX. CONCLUSION

Due to the wide spectrum of application of semantic graphs, finding similarity within and between them also has significant utility. In this paper, we focused on extending previous structural semantic similarity algorithms with functionality aimed at aiding analysts in the development of new intelligence problem decompositions. The resulting extended SSDM (or SSDM+) algorithm will be incorporated in a tool that finds fragments of existing graphs, which could be relevant to the problem being decomposed. This not only saves analyst time by allowing them to reuse relevant pieces from existing decompositions, but also allows the exploration of options for completing the decompositions in ways that might not have been previously considered. The modifications we made to the SSDM algorithm allow it to take better advantage of the data stored in semantic graphs rather than merely using structural (physical) convergences. By providing users control (via weights) over the various types of convergences - physical, literal, and ontological - SSDM+ becomes easily customizable to varying datasets. The extended algorithm also produces results that better match human intuitions on similarity. In the future, we plan to develop a more rigorous test to quantify these improvements and to speed up the off-line (tree) generation of similarity traces.

REFERENCES

- [1] T. Berners-Lee, M. Handler, & O. Lassila, "The semantic web", *Scientific American*, May 2008. [Online]. Available: http://www.ds3web.it/miscellanea/the_semantic_web.pdf
- [2] G. Jeh & J. Widom, "SimRank: a measure of structural-context similarity", in *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, ser. KDD '02 New York, Ny: ACM, 2002, pp. 538-543. [Online]. Available: <http://doi.acm.org/10.1145/77504/775126>
- [3] A. Islam & D. Inkpen, "Semantic text similarity using corpus-based word similarity and string similarity", *ACM Transactions on Knowledge Discovery from Data*, vol. 2 no. 2 pp. 1-25, July 2008. [Online]. Available: <http://doi.acm.org/10.1145/1376815.1376819>
- [4] R. Hartley & A. Zisserman, *Multiple view Geometry in Computer Vision*. Cambridge University Press, 2003. [Online]. Available: <http://books.google.com/books?id=si3R3Pfa98QC>
- [5] D. Fogaras & B. Racz, "Scaling link-based similarity search", in *Proceedings of the 14th international conference on World Wide Web*, ser. WWW '05. New York, NY: ACM, 2005, pp. 641-650. [Online]. Available: <http://doi.acm.org/10.1145/1060745.1060839>
- [6] W. H. Gomaa, "A Survey of Text Similarity Approaches", *International Journal of Computer Applications*, vol 68, no. 13, 2013. doi:10.1.1.403.5446
- [7] C. Olsson, P. Petrov, J. Sherman, A. Perez-Lopez. "Finding and Explaining Similarities in Linked Data", *Semantic Technology for Intelligence, Defense, and Security (STIDS 2011)*, Fairfax, Virginia, November 2011.
- [8] B. Gallagher "Matching Structure and Semantics: A Survey on Graph-Based Pattern Matching", *American Association for Artificial Intelligence*, 2006, [Online]. Available: <http://www.aaai.org/Papers/Symposia/Fall/2006/FS-06-02/FS06-02-007.pdf>