Incremental Graph Matching for Situation Awareness

Adam Stotz
CUBRC Inc.
4455 Genesee St.
Buffalo, NY 14225, U.S.A.
stotz@cubrc.org

Rakesh Nagi, Moises Sudit
Dept. of Industrial & Systems Engineering
Center for Multisource Information Fusion
University at Buffalo (SUNY)
Buffalo, NY 14260, U.S.A.
{nagi, sudit}@buffalo.edu

Abstract – In this paper, an incremental subgraph matching problem is introduced as an enhancement to a batched inexact subgraph isomorphism for situation assessment in higher levels of data fusion. The procedure is shown to be a bounded incremental algorithm, meaning that its runtime is a function of the size of the change in the data graph. Solution quality results are shown to be equal to that of TruST [5] with large improvements in runtime for graphs even in the size range of thousands of nodes. This new enhancement allows subgraph isomorphism procedures to be applied to new types of fusion problems.

Keywords: Graph Matching, Situation Assessment.

1 Introduction

The JDL Model defines L2 Fusion as Situation Assessment, where Situations are often times defined as the state of objects and the relationships between those objects. An increasing number of graph analytical procedures are being applied toward the situation assessment problem today. Graphs tend to be a natural way of representing concepts (including objects) and relationships between those concepts. Humans perform fusion on a daily basis and decision makers in particular, during their decision making process, naturally tend towards representing their knowledge as graphs. Graphs then become a well-suited form for Expressing and Estimating situational content. In this paper we will consider Directed Attributed Relational Graphs (DARGs), which extend the definition of simple graphs to include both vertex attributes and edge attributes. A DARG $G$ is denoted as $G = \{V, E, A_v, A_e\}$ where $V$ is a set of vertices, $E$ is a set of edges, $A_v$ is the finite set of vertex attributes and $A_e$ is the set of edge attributes.

Graph Matching, used synonymously with Subgraph Isomorphism in this paper, is a graph analytical procedure aimed at finding occurrences of a pattern or Template Graph (TG) within a usually larger corpus of evidence or Data Graph (DG). Either of these graphs can be generated by a human or computer, however in normal practice, it is usually the human defining the Template Graph to be matched into a computer generated Data Graph. These patterns or Template Graphs represent the expression by an end user of the Graph Matching system, of a set of related concepts forming something determined to be of interest in a large volume of data. Finding occurrences of a pattern in large volumes of data is a daunting task for a human to perform manually because of the exponential set of possible mappings between the TG and DG and the rate at which humans can generate and evaluate these possible mappings.

Finding occurrences of a Template Graph $G_1 = \{V_1, E_1, A_{V_1}, A_{E_1}\}$ within a Data Graph $G_2 = \{V_2, E_2, A_{V_2}, A_{E_2}\}$ can be stated as finding a subgraph $G_1$ within a graph $G_2$. $G_1$ is said to be a subgraph of $G_2$ if and only if $V_1 \subseteq V_2, E_1 \subseteq E_2, A_{V_1} \subseteq A_{V_2}, A_{E_1} \subseteq A_{E_2}$. This type of matching is termed Subgraph Isomorphism [1].

Isomorphism finds topologically equivalent occurrences of patterns in graphs, different from homeomorphism, which finds occurrences, which are semantically equivalent, and may or may not be topologically equivalent. When patterns are defined to uncover occurrences of abstract concepts such as situations, semantic equivalence becomes important to distinguish between possibly many different syntactic forms of the pattern in data. Semantic graph matching is not within the scope of this paper (see [2]).

Subgraph Isomorphism exists in the exact and inexact varieties. Exact Subgraph Isomorphism finds $G_1$ as a subgraph of $G_2$, say $G_1^2$, with the additional constraint that $G_1 = G_2$. In reality, Data Graphs are often littered with missing or incorrect data, and Template Graphs are often too complex to perfectly model what is looking to be found in the DG. These real world problems make applying Graph Matching techniques of the inexact variety more likely to yield better results due to their much less sensitive nature to these topological inconsistencies. We will focus the rest of this paper on the inexact variety for this reason.

2 Background

2.1 Graph Matching Review

Graph based matching techniques have been around for a number of decades [3]. The eighties brought in a flurry of work. Increased computational power of recent computers has spurred renewed interest towards graph matching techniques. Graph based pattern recognition or graph matching is the process of finding a correspondence between the nodes and the edges of two graphs that satisfy some constraints ensuring semantic and syntactic
consistency between the two. In this section, we will present a brief literature review in the field of graph matching and pattern recognition techniques.

As mentioned in Section 1, graph matching techniques are divided into two broad categories: (1) the exact graph matching method that requires stringent correspondence among the graphs to be matched, and (2) the inexact graph matching method, where two graphs can be compared even though they are semantically or topologically different.

Exact graph matching is defined as a mapping between the nodes of the two graphs, with the edge-preserving characteristic. It means that if two nodes in one graph share an edge, then the two nodes in the second graph also share an edge. The most stringent form of exact graph matching is the “graph isomorphism”. In this case, the mapping must be bijective, which means that there is a one-to-one correspondence between the nodes and edges of each graph. A bit weaker form of exact graph matching is the “subgraph isomorphism” [1]. Here, a subgraph of one graph is checked to be isomorphic to another graph. If the surjective condition (“onto relationship”) is dropped then we have “monomorphism”. In monomorphism, node-to-node correspondence exists between both graphs, but the edge correspondence need not exist. In a still weaker kind of matching, called “homomorphism”, the condition that, nodes in the first graph are to be mapped to distinct nodes of the other, is dropped. The matching problems mentioned above are all NP-complete, except if we have an attributed graph in which the nodes are guaranteed to have distinct attributes, and then exact graph matching is polynomial.

In inexact graph matching [4] some of the stringent conditions imposed by exact matching are relaxed to accommodate the disturbances in the system. In this case, two nodes that do not satisfy the edge preserving condition can be mapped, but there is a cost assigned to this mapping. Also in the attributed graph case two nodes may be mapped even if $A_{1} \neq A_{2}$. The inexact graph matching algorithm works with an objective function of minimizing the mapping or matching cost. Optimal inexact matching algorithms always find an exact solution, if one exists.

Approximate or suboptimal matching algorithms find local minima, which might (or might not) be far from the global minima. A detailed review of these advances can be found in [5].

2.2 Past work on Static Graph Matching

Our past work has focused on a Truncated Search Tree (TruST) approach to the inexact subgraph isomorphism problem on DARGS [5]. TruST is a breadth-first greedy state space search heuristic to solve this NP-Hard problem. TruST is akin to the beam search algorithm, which is a variation of best-first search. Like beam search, it is space-bounded as a function of $x$, where $x$ is a fixed number – the “beam width.” At each branch depth in the search tree, the heuristic further branches from only the first $x$ most promising nodes to form the next depth level of the tree. While this limits the solution search space of the problem to improve time performance, it is not guaranteed to yield an optimal solution. However, the optimality gap of the reached sub-optimal solution is often within the system’s requirements for solution quality and through the control of the truncation parameters one can obtain various efficiency tradeoffs. The various parameters of the heuristic (including $K_{0}$ and $K_{1}$ shown in Figure 1) are not important in the sense of the rest of our discussion in this paper.

![Figure 1 - TruST Search Tree Example](image)

The TruST search tree is shown in Figure 1. This tree is generated dynamically during the heuristic’s execution and initially consists of only the root ($\emptyset \rightarrow \emptyset$) node. Within each iteration of the algorithm, the current partial match (represented by a node in the search tree) is expanded by one more mapping of a TG node to a DG node. Each node of depth $n$ ($n = 0,1,2,...$) in the search tree represents a unique mapping of $n$ TG nodes to $n$ DG nodes. The algorithm uses a breadth-first search strategy meaning that all nodes are processed at depth $n$ in the search tree before any at depth $n + 1$.

2.3 Incremental Graph Analytics

Graph Analytical Algorithms in general suffer from performance problems making their use most normally restricted to forensic types of applications or retrospective analysis. They are also inherently a static process, meaning that if a result set is obtained at time $t_{0}$, there is a change made to the DG at time $t_{1}$, then to have this change reflected in the results at time $t_{2}$ ($t_{2} > t_{1} > t_{0}$), the entire algorithm must be re-run. In environments where the data graph is undergoing constant change, which is becoming ever more frequent with the increase in sensing capability, graph analytical technologies become increasingly less attractive as a solution.

In the role of technology as applied to Situation Assessment problems, Graph Matching as a static graph analytical algorithm suffers from these same application limiting performance characteristics. In tactical fusion applications and even operational and strategic fusion applications
requiring high up-tempo, waiting even minutes for results of a single template is detrimental. A situational assessment system will typically have numerous template graphs to match. As this number increases, the time to retrieve the results will grow best-case linearly also, assuming independent runs of template graphs.

Dynamic Graph Algorithms have been defined by Eppstein et al. [6] to be those algorithms “that maintain some property of a changing graph more efficiently than recomputation from scratch after each change.” The work then goes on to define partially dynamic and fully dynamic algorithms which allow only edge insertions, and both edge insertions and deletions, respectively. A fully dynamic graph algorithm typically is composed of two largely independent sub-algorithms to operate on an edge addition versus an edge deletion. The efficiencies to be gained differ in the case of these two graph operations, requiring different algorithms, and even resulting in different time complexities between the two. As such, the time complexity of a fully dynamic graph algorithm will usually be given as the time complexity under the addition operation and the time complexity under the deletion operation.

It is important to define the term dynamic graph. The definition of dynamic graphs and dynamic graph algorithms as used in this paper is in the likes of Eppstein – that is, dynamic topological changes in a graph structure. This is not to be confused with other definitions of dynamic graphs [7] which define the term in the sense of modeling complex dynamic systems as directed graphs of vertices (subsystems) and edges (interconnections). These are distinct concepts, the systems modeling concept focused on finding equilibrium and stability for control problem, versus our concept as representing assertions of concepts. Edges are not static existent graph components whose quantitative value is continuous and time-dependent. Rather, the topological existence of the edge itself is dynamic. Edge attributes which are discretely dynamic can be modeled as the removal of the edge with the old value, and addition of the edge with the new value.

In the following we propose a new incremental, inexact subgraph isomorphism search procedure that is suitable for situational understanding under streaming data. The advantages of this approach over a static graph search like TruST, is in the types of applications it can serve due to the nature of the temporal performance characteristics of incremental graph analytics over static graph analytics.

2.4 Incremental Graph Matching and Situation Awareness

There are a number of Information Fusion (IF) Reference Models [2] which intend to provide definitions for IF concepts such as Situation, Situation Assessment, Situation Awareness, etc. To avoid further philosophical debate in this paper, we will assume that Graph Matching technologies relate to Situation Awareness because they are useful in helping a user derive information from raw data. This derived information (a subgraph isomorphism), if pertinent to a user’s decision space, can be considered as a (or component of a) situation assessment product, as situation assessment is commonly understood.

The raw data in this sense is a data graph containing observables in which specific combinations (subgraphs) of nodes, edges, and attributes can have decision space relevant meaning. The relevance of a specific subgraph is expressed as a Template Graph (Target Graph), and the role of an inexact subgraph isomorphism heuristic is to search the typically large Data Graph for the typically much smaller subgraphs which match, to some degree, that expressed relevance.

Observables frequently change and decision space relevance as well, in common fusion applications. Graphs representing these observables (data graph) and decision space relevance (template graph) will then change frequently as well. As the frequency of change increases, subgraph matchings can become outdated and lose their meaning as any stale estimates would in a changing environment. The incremental graph matching approach discussed in this paper addresses this problem by ensuring that matches remain current in a highly dynamic environment. It allows the expressive power of graph representations and the rigorous graph-theoretical analysis to be applied to these problems while minimizing their major traditional disadvantage of computational runtime.

3 Incremental Subgraph Search

In most fusion applications there is a time interval in which a given estimate will remain valid. Environments change dynamically, the frequency of which partly determines this time interval. As the temporal lag increases between the production of an estimate and the input of the observations to the estimation process, the less applicable that process is for current state estimation. A design requirement of the incremental search procedure is that it should be consistent with the rate of change.

We continue to assume the Template Graph in a graph matching process represents the Entities of Interest. These entities can be objects, relationships, activities, etc. (any type of an entity) with the underlying assumption of common data semantics between the data graph representation of those concepts and the template graph representation of those concepts. For example, if an activity is a single node in the data graph, it should not be represented as multiple related nodes in the template graph. This is consistent with the basic assumption of isomorphism.
Our procedure takes as input the assigned utility of the streaming changes. This is a quantitative utility value (match value), a measure of how closely the template nodes and edges match the data graph nodes and edges for the particular mappings of the given result, based on their attributes without taking into account topology. This type of utility assessment is assumed to be provided or calculated as a preprocessing step to the search heuristic.

### 3.1 Technical Approach

Figure 2 depicts the high-level technical approach as it compares to that of non-incremental isomorphism. A common input to both incremental and non-incremental matching is the template graph to be found within the (usually much larger) data graph. Contrary between the two approaches are the other required inputs; in the non-incremental matching sense, the only other input needed is the entirety of the data graph, and from the incremental perspective only the Data Graph (DG) change is needed as input along with an initial matching result valid for the data graph as it existed just before the incremental DG change.

![Figure 2 – Incremental Graph Search Inputs and Outputs](image)

The updated match result generated by incremental search is grounded on heuristics so only bounds on quality can be guaranteed. What will be discussed and focused on is the comparison of these results over a time series of incremental changes to the data graph versus TruST results for the same end result data graph. Both tools will be provided with an identical template graph.

The heuristic implemented within the incremental approach is scientifically based on the theory developed previously for TruST and will be investigated further in this Section. TruST can produce time efficient results by parametrically bounding the state space it searches. Figure 1 was a representation of that state space as the search tree generated within TruST. Still, to provide heuristically good results, this search tree can still grow to be large and is entirely regenerated for each run of the TruST heuristic. In applications where the data graph is frequently changing, these small incremental changes result in massively redundant computations across scheduled batch invocations of TruST. We hope to eliminate these redundant computations, while leveraging the search tree research proven to be advantageous in heuristically solving the inexact subgraph isomorphism problem. Figure 3 illustrates the basic idea. Note that there is no significance to the branches highlighted.

### 3.2 Initialization Phase – Reconstruction of State Space Search Tree

Prior to the processing of the data graph change stream, preprocessing of the initial match result is needed to ensure proper execution during the heuristic when determining the affected state space (see [2]). The fundamental concept of our incremental search is based in the same state space search tree used within TruST. It is not a strong assumption that the original TruST search tree is available. This simply makes it compatible with any subgraph isomorphism heuristic or algorithm and we can show that necessary branches within the State Space Search Tree can be reconstructed from a single or set of match results.

A branch within the State Space Search Tree can be regenerated from any match result. This is true even of results that do not come from TruST. Consider Figure 4 – in this example there are four match results given \( \{R^t_1, R^t_2, R^t_3, R^t_4\} \) with corresponding match result scores \( R^t_1 = .700, R^t_2 = .683, \ldots \) taken or computed as the mean of the single map scores \( h^k_{ij} \in R_k \). Each \( h^k_{ij} = h(v^T_i, v^D_j), \forall (v^T_i, v^D_j) \in R_k \). The scoring is consistent with that of TruST, so in cases where \( R^t \) is a product of TruST, these 1-hop neighborhood computations do not need to be recomputed. In cases where they must, ISIS still remains theoretically bounded in the incremental algorithm sense,
since the number of \( h \) computations given by \( |V^T||R^T| \) does not scale with the size of the data graph.

Ensuring the reconstructed search tree is consistent with the TruST methodology of its generation implies an assumption regarding the ordering of node mappings within each match result, \( R^T_i \in R^T \). TruST is breadth-first greedy, and therefore match results, as leaves of the search tree, must reflect this nature. Considering \( R^T_i \) generically, not as a TruST product, we must order the node mappings within the result in a greedy fashion while respecting the topology of the template graph with a descending adjacency sort function.

The descending adjacency sort function sorts the result node mappings in descending map value order, but under the condition that template node \( v^T_i \in m_i \) of node mapping \( m_i \in R^T_i \), \( \forall i, k, i \neq 1 \) in the sorted list is adjacent to the subgraph induced from \( \bigcup_{j=1}^i v^T_j \). Figure 5 provides an example of the fallacy of not enforcing this adjacency condition when performing the sort. In example 1, the adjacency condition holds fine; E.g. \( v^T_1 \) is the first list item so by default is satisfied, \( v^T_2 \) is adjacent to \( \{v^T_1\} \), \( v^T_3 \) is adjacent to \( \{v^T_1, v^T_2\} \), and \( v^T_4 \) is adjacent to \( \{v^T_1, v^T_2, v^T_3\} \). In example 2 however, \( v^T_1 \) is not adjacent to \( v^T_1 \). Reconstructing the branch of the search tree corresponding to this result would lead to an inconsistency between the reconstructed tree and the tree TruST would have generated. Within the TruST branching strategy, available node mappings to grow a partial result with is chosen from the available mappings satisfying this same adjacency criteria. Note that in the corrected sorted list shown in the figure and labeled as “Adjacency enforced”, node map scores are not necessarily monotonically decreasing.

From the adjacency sorted results, the state space search tree can now be reconstructed quite easily. Referring to Figure 4, the tree is built bottom-up. Starting with the result \( R^T_1 \) at tree depth level \( n \) (as the leaf of a tree branch), the parent tree node at level \( n - 1 \) is constructed by simply taking the first \( n - 1 \) mappings from the adjacency sorted mapping list. The parent parent's node can then be constructed by taking the first \( n - 2 \) mappings and so on. Reconstructed tree branches merge when two or more results contain a common mapping at the same index of the sorted mappings list. This is the case of \( R^T_1 \) and \( R^T_2 \) in the example.

Figure 6 shows the final reconstructed State Space Search Tree. The reconstruction process is actually not lossless. In the TruST branching strategy, pruning is performed to maintain efficiency. These pruned branches never lead to results, and therefore can never be reconstructed without modifying TruST to cache this type of information. From the incremental search perspective, this missing information is not a dependency and may never even exist when considering other approaches to the subgraph isomorphism problem other than TruST. A depiction of the types of pruned search tree branches that are missing in the reconstructed search tree is provided in Figure 7. This information is not so important because it does not affect the modification of the current result set when data graph changes are processed. In the worst case, these branches will be recomputed and re-pruned within the incremental search heuristic where control is available to cache pruned branches if necessary.
Each data graph change notification is one of the following types:

- **Nodes Added** – One or more nodes have been added to the data graph.
- **Edges Added** – One or more edges have been added to the data graph.
- **Components Added** – One or more nodes and edges have been added to the data graph.
- **Nodes Removed** – One or more nodes have been removed from the data graph.
- **Edges Removed** – One or more edges have been removed from the data graph.
- **Components Removed** – One or more nodes and edges have been removed from the data graph.

A graph component is simply a node or edge. Components Added/Removed captures all other change types, but the reasoning behind including special case change types is for special case processing that can be more efficient. Addition change types will be considered in this work, while removal change types will be left for future work. Both types of data graph changes are important, but insert and update transaction types are logical to investigate first seeing as you only can delete what has been added at some point in the past. Also, many data bases which act as historical persistent data stores do not delete information intentionally as part of the data policy governing its use. These data bases become data graphs to the heuristic and the transactions adding to them then become the data graph change stream.

### 3.4 Steps of the Incremental Search

The first step in the incremental search process when a DG change notification is received is to determine the affected nodes of the data graph, \( V^{AFF} \equiv \{v^D_1, v^D_2, ..., v^D_n\} \). We determine that a modified vertex \( u \) in \( G + \delta \) is one in which it is modified by the incremental change \( \delta \) such that \( u \) is inserted or deleted, had attributes modified, or had any of these operations on an incident edge." \( V^{AFF} \) is consistent with this definition with the inclusion that attribute modification is captured through the removal and addition of the vertex or edge owning the attribute. A single graph notification containing the component additions \( \delta \equiv \delta^V \cup \delta^E \) results in \( V^{AFF} = v^D_r \cup v^D_i \cup v^D_j \), \( \forall v^D_r \in \delta^V \), \( \forall e(v^D_i, v^D_j) \in \delta^E \).

The next step in the search process is to actually update the result set from the previous time increment \((R^{t-1})\) to be current with the changes that have been applied to the data graph \((\Delta^t)\). This result updating is achieved through the determination and recalculation of the affected state space search tree \((T^{AFF} \subseteq T)\) where \( T^{AFF} \) is the set of affected tree nodes within the search tree \( T \). Since we are closely aligned with TruST, it carries over in determining \( T^{AFF} \) and calculating \( R^t \) by recomputing \( T^{AFF} \).
In the worst case, $T^{AFF} = T$, where there is no computational performance benefits over running TruST on the newly updated data graph $(DG_t^r)$ at time $t$. We can prove however that under the assumption of node/edge addition the probability of encountering that worst case scenario decreases with increasing result quality. Also, under node/edge addition or removal, the probability also decreases with a decrease in the density of the graph.

As a precursor to determining $T^{AFF}$, we have previously described the calculation of $V^{AFF}$, and the calculation of $\Delta HT$. Just as affected data graph nodes determine the changes to the 1-Hop Neighborhood Scores, changes in 1-Hop Neighborhood Scores form the basis for determining affected search tree nodes.

<table>
<thead>
<tr>
<th>$v^r_0$</th>
<th>$v^r_1$</th>
<th>$v^r_2$</th>
<th>$v^r_3$</th>
<th>$v^r_4$</th>
<th>$v^r_5$</th>
<th>$v^r_6$</th>
</tr>
</thead>
<tbody>
<tr>
<td>.85  (+.30)</td>
<td>.76  (+.03)</td>
<td>.56  (+.03)</td>
<td>.50  (+.00)</td>
<td>.45  (+.00)</td>
<td>.01  (+.20)</td>
<td>---</td>
</tr>
</tbody>
</table>

Figure 4 – Example HT Values

Figure 8 shows some of the sample values in the $\Delta HT$ table sorted into a list by decreasing $h$ value. The value shown in parentheses within each list cell is the change between the current $(h^r)$ value and the value before the change notification $(h^{r-1})$.

4 Results

To show the quality of results in relation to TruST, 500 results were collected across 10 independent random graph generations to ensure a fair statistical test. Each sample data graph was a randomly generated topology with randomly selected attributes consisting of twice as many edges as nodes, keeping the graph density fixed. Each sample template graph was randomly generated in a similar manner, but much smaller in size – 5 nodes and 10 edges. TruST was initialized with identical heuristic parameters for all tests: $\alpha = .5, k_0 = 50, k_1 = 10, \beta = 50, \delta = 10000$. The large $\delta$ value ensures the branching procedure recurses deep enough to find complete matches if they exist.

4.1 Solution Quality

To test about the equality of means, parametric approaches such as the two sample t-test can be used under the assumption of a normal distribution and equal variances. If these assumptions cannot be made, other non-parametric approaches exist to test about the equality of medians instead of means such as the Mann-Whitney test. Our data failed the test for normal distribution, and the results from Levene’s test provided equal variances. Thus, Mann-Whitney test about the medians must be used instead. In the test, the null hypothesis of having equal medians fails to be rejected meaning that with 99.9% confidence, incremental search and TruST results have equal medians of result scores.

4.2 Solution Timeliness

The advantage of incremental over non-incremental search is in the runtime required to obtain current match results. To compare the temporal runtime benefits, an initial random template graph was generated as mentioned before. For statistical fairness, 3 trials of 100 random data graphs were generated with sizes $n=100$ to $n=10000$. These parameters provide for a wide range of data graph sizes to test the scaling of both incremental and non-incremental runtimes in addition to the scaling of the difference in runtimes between them. There is a single change generated for each generated data graph of size 100 making the size of the final data graph equal to the size of the next randomly generated initial data graph.

Figure 10 shows the runtime of TruST as the data graph grows from 100 nodes to 10,000 nodes. From the trend line displayed, it can be seen that the runtime grows non-linearly for larger graphs. Running ISIS on these same graphs, produces the runtimes shown in Figure 9. As the size of the data graph grows, the runtime benefits of ISIS become increasingly profound. Having a consistent runtime
of less than one second under the condition of a growing data graph empirically suggests that our incremental search algorithm may be a bounded, meaning that its runtime is invariant to the size of the underlying data graph.

5 Conclusions

In Situation Assessment, the state of objects and the relationships between those objects can be conveniently represented as Graphs. Graph Matching is a graph analytical procedure aimed at finding occurrences of a pattern or Template Graph (TG) within a usually larger corpus of evidence or Data Graph (DG). Past work has mainly considered the data graph to be static, where in fact, the state of the data graph is constantly evolving as new evidence comes in. Rather than repeat search of the template graph at each instance of change, this paper proposed an incremental subgraph isomorphism approach. The algorithm strongly draws from our previous state space search approach, TruST. We have shown that the incremental search procedure generates solutions of the same (or similar) quality in significantly lower computational times. We are considering semantic enhancements to this class of graph isomorphism problems. We hope that vocabulary heterogeneity and uncertainty are well suited challenges to be mitigated with the semantic enhancements.

References


