Inexact Attributed Subgraph Matching

Thomas K. Tu, Jacob D. Moorman, Dominic Yang, Qinyi Chen, and Andrea L. Bertozzi
Department of Mathematics, UCLA, Los Angeles, CA 90095
thomastu@math.ucla.edu, jacob@moorman.me, domyang@math.ucla.edu,
qinyic@mit.edu, bertozzi@math.ucla.edu

***Abstract*—We present an approach for inexact subgraph matching on attributed graphs optimizing the graph edit distance. By combining lower bounds on the cost of individual assignments, we obtain a heuristic for a backtracking tree search to identify optimal solutions. We evaluate our algorithm on a knowledge graph dataset derived from real-world data, and analyze the space of optimal solutions.**

I. Introduction

Subgraph matching involves searching for a prescribed *template* graph as a subgraph of a *world* graph. One may want to find one occurrence, all occurrences, or some other summary of the solution space, depending on the application [1]. In applications where the graphs have attributes on the nodes and edges, it is common to search for subgraphs of the world which only approximately match the template. This is called *inexact attributed subgraph matching*.

Definition I.1 (Attributed Graph)**.** *An attributed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{L}, \mathcal{A})$ consists of a set of nodes $\mathcal{V}$, a set of edges $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$, a set of attributes $\mathcal{A}$, and a map $\mathcal{L} : \mathcal{V} \cup \mathcal{E} \to \mathcal{A}$ from nodes and edges to their attributes.*

Given two attributed graphs, a template $\mathcal{G}_t = (\mathcal{V}_t, \mathcal{E}_t, \mathcal{L}_t, \mathcal{A})$ and a world $\mathcal{G}_w = (\mathcal{V}_w, \mathcal{E}_w, \mathcal{L}_w, \mathcal{A})$, we are interested in one-to-one maps $f : \mathcal{V}_t \to \mathcal{V}_w$ where the induced subgraph of the image is similar to the template. Such a map f is called an *inexact match* of $\mathcal{G}_t$. The cost metric used to measure similarity between the template and the image of f is denoted $C(f; \mathcal{G}_t, \mathcal{G}_w)$ and is described in Section II-A.

The process of finding the map $f^\star$ which minimizes $C(f; \mathcal{G}_t, \mathcal{G}_w)$ is called inexact *attributed subgraph isomorphism (ASI)*, while the process of finding all maps f with distance at most ϵ is called inexact *attributed subgraph matching (ASM)*. Finding the k most similar maps is called top-k inexact ASM.

In the remainder of this section, we discuss existing approaches to ASI/ASM and inexact ASI/ASM, as well as our contributions. In Section II, we describe our approach to solving inexact ASI, inexact ASM, and top-k inexact ASM. In Section III, we evaluate our methods on the AIDA V2.1.2 dataset.

A. Related Work

Algorithms for exact subgraph matching mostly follow one of three approaches [2], [3]: tree search [4], [5], [6], constraint propagation [7], [8], [9], [10], [11], [12], and graph indexing [13], [14], [15], [16], [17], [18].

Tree search approaches navigate the space of all possible maps from the template nodes to the world nodes, making one assignment at a time, backtracking when it becomes clear an isomorphism is impossible with the current assignment. Constraint propagation approaches follow the same tree search procedure while additionally tracking which world nodes are candidates for which template nodes in a compatibility matrix. By repeatedly applying local constraints, they reduce the candidates before each assignment.

Graph indexing is typically used for graph database search based on a subgraph query. To accelerate searching, one constructs indexes based on characteristic substructures of the template. A Cartesian product on the results of these indexes identifies possible matches. A verification step follows to check which retrieved graphs fully match the query; this typically involves running another subgraph isomorphism algorithm.

Algorithms for inexact subgraph matching are more diverse than those for exact subgraph matching. Different applications and research areas define the inexact subgraph matching problem in different ways. Some algorithms take tree search, constraint propagation, or graph indexing approaches similar to that seen in exact subgraph matching [19], [20], [21], [22], while other algorithms relax the discrete optimization problem into a continuous one in order to apply traditional continuous optimization techniques such as gradient descent [23].

B. Contributions

We introduce algorithms for inexact attributed subgraph isomorphism and matching to find optimal sub-

978-1-7281-6251-5/20/$31.00 ©2020 IEEE

graphs as measured by graph edit distance. We show results for noisy queries on knowledge graphs.

II. Algorithm

In Section II-A, we define the cost metric $C(f;\mathcal{G}_t,\mathcal{G}_w)$ that will be minimized in inexact ASI/ASM. In Sections II-B and II-C, we explain how to compute lower bounds on $C(f;\mathcal{G}_t,\mathcal{G}_w)$ under the constraint $f(t_c)=w_c$ for each $t_c\in\mathcal{V}_t$ and $w_c\in\mathcal{V}_w$. Using these constrained lower bounds, in Section II-D we describe a tree search procedure for inexact ASI/ASM.

A. Graph Edit Distance Based Cost Metric

The graph edit distance [24], [25] is a measure of the distance between two graphs, as defined by the total cost of the cheapest sequence of edits which transform one graph into another. The six possible edits usually considered are node addition, node deletion, node substitution, edge addition, edge deletion, and edge substitution. Here, we consider an cost metric using only the latter four edits.

The node substitution costs are measured by a user-defined function $D_{\mathcal{V}}:(\mathcal{V}_t\times\mathcal{V}_w)\to\mathbb{R}^+$ that typically compares the labels of the nodes. A common node substitution function is

$$D_{\mathcal{V}}(v_t,v_w)=\mathbb{1}\left[\mathcal{L}_t(v_t)\neq\mathcal{L}_w(v_w)\right]$$

so that $\sum_{t\in\mathcal{V}_t}D_{\mathcal{V}}(t,f(t))$ counts the number of node assignments which do not preserve the node label. If the node labels belong to a normed vector space, the corresponding norm can be used

$$D_{\mathcal{V}}(v_t,v_w)=\left\|\mathcal{L}_t(v_t)-\mathcal{L}_w(v_w)\right\|.$$

Likewise, the edge substitution, addition and deletion costs are measured by user-defined functions $D:(\mathcal{E}_t\times\mathcal{E}_w)\to\mathbb{R}^+$, $D^+:\mathcal{E}_w\to\mathbb{R}^+$ and $D^-:\mathcal{E}_t\to\mathbb{R}^+$. The edge related cost functions are combined into a single function $D_{\mathcal{E}}$ for brevity in Equation (1). For convenience, we use the shorthand $f((t_1,t_2))=(f(t_1),f(t_2))$ so that $f(e)$ is well-defined.

$$D_{\mathcal{E}}(e,f(e))=\begin{cases}D(e,f(e)) & e\in\mathcal{E}_t,f(e)\in\mathcal{E}_w\\ D^-(e) & e\in\mathcal{E}_t,f(e)\notin\mathcal{E}_w\\ D^+(f(e)) & e\notin\mathcal{E}_t,f(e)\in\mathcal{E}_w\\ 0 & e\notin\mathcal{E}_t,f(e)\notin\mathcal{E}_w.\end{cases}\tag{1}$$

Given a template $\mathcal{G}_t$, a world $\mathcal{G}_w$, and a map $f:\mathcal{V}_t\to\mathcal{V}_w$, the cost metric $C(f;\mathcal{G}_t,\mathcal{G}_w)$ is defined as

$$C(f;\mathcal{G}_t,\mathcal{G}_w)=\sum_{t\in\mathcal{V}_t}D_{\mathcal{V}}(t,f(t))+\sum_{e\in\mathcal{V}_t\times\mathcal{V}_t}D_{\mathcal{E}}(e,f(e)).\tag{2}$$

B. Cost Bounds

We now discuss lower bounds on the cost metric. $C(f;\mathcal{G}_t,\mathcal{G}_w)$ can be decomposed into local costs $L(t,f(t);\mathcal{G}_t,\mathcal{G}_w,f)$ related to each template node t and matching world node $f(t)$

$$C(f;\mathcal{G}_t,\mathcal{G}_w)=\sum_{t\in\mathcal{V}_t}L(t,f(t);\mathcal{G}_t,\mathcal{G}_w,f).$$

We use decompositions where the local cost L has the form

$$\begin{aligned}L(t,w;\mathcal{G}_t,\mathcal{G}_w,f)&=D_{\mathcal{V}}(t,w)\\&+\frac{1}{2}\sum_{t_o\in\mathcal{V}_t}\Big[D_{\mathcal{E}}\big((t,t_o),(w,f(t_o))\big)\\&\qquad+D_{\mathcal{E}}\big((t_o,t),(f(t_o),w)\big)\Big].\end{aligned}\tag{3}$$

This way, the local cost $L(t,w;\mathcal{G}_t,\mathcal{G}_w,f)$ captures the cost incurred by assigning world node w to template node t and half of the cost of assigning the edges incident to w to those incident to t. Edge costs are halved so that the local costs sum to the correct total cost $C(f;\mathcal{G}_t,\mathcal{G}_w)$; otherwise, edge costs would be counted twice, once for each endpoint. If the edge addition cost were neglected (i.e. $D^+\equiv 0$), the summation $\sum_{t_o\in\mathcal{V}_t}$ reduces to $\sum_{t_o\in\mathcal{N}_t}$, where $\mathcal{N}_t$ is the set of neighbors of t. One often useful variation is to alter these factors independently for each edge to place more importance on certain nodes; we omit this discussion for now, but refer to Appendix A for the details.

To bound the full cost, we start by bounding the local cost $L(t,w;\mathcal{G}_t,\mathcal{G}_w,f)$ from below by $B(t,w;\mathcal{G}_t,\mathcal{G}_w)$ defined as follows

$$\begin{aligned}B(t,w;\mathcal{G}_t,\mathcal{G}_w)&:=D_{\mathcal{V}}(t,w)\\&+\frac{1}{2}\sum_{t_o\in\mathcal{V}_t}\min_{w_o\in\mathcal{V}_w}\Big(D_{\mathcal{E}}\big((t,t_o),(w,w_o)\big)\\&\qquad+D_{\mathcal{E}}\big((t_o,t),(w_o,w)\big)\Big).\end{aligned}\tag{4}$$

Within the context of inexact ASI/ASM, the set $\mathcal{V}_w$ iterated over in the minimization can often be reduced for practical purposes; we refer to Appendix B for the details.

We can extend this bound on the local cost to a naive bound on the full cost

$$\begin{aligned}C(f;\mathcal{G}_t,\mathcal{G}_w)&\geq\sum_{t\in\mathcal{V}_t}B(t,f(t);\mathcal{G}_t,\mathcal{G}_w)\\&\geq\sum_{t\in\mathcal{V}_t}\min_{w\in\mathcal{V}_w}B(t,w;\mathcal{G}_t,\mathcal{G}_w)\end{aligned}\tag{5}$$

which is sufficient for some applications. Alternatively, we can compute a tighter lower bound by leveraging the fact that f must be one-to-one (1-1).

$$G(f;\mathcal{G}_t,\mathcal{G}_w) := \min_{\substack{g:\mathcal{V}_t\to\mathcal{V}_w \\ \text{s.t. } g \text{ is 1-1}}} \sum_{t\in\mathcal{V}_t} B(t,f(t);\mathcal{G}_t,\mathcal{G}_w) \quad (6)$$

$$C(f;\mathcal{G}_t,\mathcal{G}_w) \geq G(f;\mathcal{G}_t,\mathcal{G}_w) \quad (7)$$

We refer to G as the global cost bound. Computing this bound is equivalent to solving a rectangular linear assignment problem (LAP) of size $|\mathcal{V}_t| \times |\mathcal{V}_w|$. Many algorithms exist to solve the LAP and its rectangular variant. The most notable ones include the Hungarian algorithm [26], the Munkres algorithm [27], the Jonker-Volgenant (JV) algorithm [28], and the auction algorithm [29]. In our implementation, we apply a modified JV algorithm [30], which is designed to efficiently solve rectangular linear sum assignment problems via a shortest augmenting path approach. The time complexity of a JV solver is typically $\mathcal{O}(|\mathcal{V}_t|\,|\mathcal{V}_w|^2)$.

C. Constrained Cost Bounds

To use the lower bounds from Equations (5) and (6) in our algorithm, we must compute the bounds under the constraint $f(t_c) = w_c$ for each $t_c \in \mathcal{V}_t$ and for each $w_c \in \mathcal{V}_w$. The corresponding constrained lower bounds are

$$B(t_c,w_c;\mathcal{G}_t,\mathcal{G}_w) + \sum_{\substack{t\in\mathcal{V}_t \\ t\neq t_c}} \min_{\substack{w\in\mathcal{V}_w \\ w\neq w_c}} B(t,w;\mathcal{G}_t,\mathcal{G}_w) \quad (8)$$

for the naive bound and

$$G(t_c,w_c;\mathcal{G}_t,\mathcal{G}_w) := \min_{\substack{g:\mathcal{V}_t\to\mathcal{V}_w \\ \text{s.t. } g \text{ is 1-1} \\ \text{s.t. } g(t_c)=w_c}} \sum_{\substack{t\in\mathcal{V}_t \\ t\neq t_c}} B(t,f(t);\mathcal{G}_t,\mathcal{G}_w) \quad (9)$$

for the global cost bound.

To compute these constrained bounds, we first compute $B(t,w;\mathcal{G}_t,\mathcal{G}_w)$ for each $t \in \mathcal{V}_t$ and $w \in \mathcal{V}_w$. This takes $\mathcal{O}\left(|\mathcal{V}_t|\,|\mathcal{V}_w|\,\text{avgtime}(B)\right)$ time. The remaining calculations to compute Equation (8) for each $t_c \in \mathcal{V}_t$ and $w_c \in \mathcal{V}_w$ can be done in $\mathcal{O}(|\mathcal{V}_t|\,|\mathcal{V}_w|)$ time. Computing Equation (9) is more complicated.

To compute Equation (9) we must solve a LAP of size $(|\mathcal{V}_t|-1)\times(|\mathcal{V}_w|-1)$ for each $t_c \in \mathcal{V}_t$ and $w_c \in \mathcal{V}_w$. Naively solving each of these LAPs separately requires $\mathcal{O}(|\mathcal{V}_t|^2\,|\mathcal{V}_w|^3)$ time. However, we expect that there are more efficient ways to solve the LAPs. For example by solving each LAP in parallel or by first solving the unconstrained LAP and then updating the solution to enforce each constraint.

D. Search for Optimal Solutions

From Section II-C, we have a procedure for computing lower bounds on $C(f;\mathcal{G}_t,\mathcal{G}_w)$ under the constraint $f(t_c) = w_c$ for each $t_c \in \mathcal{V}_t$ and $w_c \in \mathcal{V}_w$. Now, we treat these lower bounds as a heuristic for performing a greedy depth first search. For each template node t, we assign $f(t) = w$ for the candidate w with the lowest bound, then recompute the bounds under that additional assignment. We assign candidates to template nodes with the fewest minimum bound candidates first, as this gives an indication of which template nodes have only a few "good" choices. After assigning all template nodes in this way, we obtain a map f.

Although f may not be the optimal map $f^\star$ that we seek, it can be used to drastically cut down on the list of possible assignments we have to consider going forward. We compute the cost of f to serve as an upper bound on the cost of the optimal map $f^\star$. We keep track of the *cost threshold* U and set it equal to the smallest cost $C_{\min}$ we have seen so far, with $f_{\min}$ the corresponding map. Using $C_{\min}$, the search space is refined by skipping assignments $f(t) = w$ which lead to lower bounds that are greater than or equal to $C_{\min}$, since in inexact ASI we are only interested in the map $f^\star$ which minimizes the cost.

After identifying a new map or eliminating all remaining possibilities due to the cost bounds, we backtrack and try assigning different world nodes to some template nodes. This is done until there are no options left to explore, at which point we have found the optimal map $f^\star = f_{\min}$.

To perform inexact ASM instead of inexact ASI, one can fix $U \equiv \epsilon$ and record all of the maps f that are observed during the search. In this context, we only skip assignments when their cost bound is strictly greater than ϵ, so that we do not accidentally skip matches whose cost is exactly ϵ.

To perform top-k inexact ASM, keep track of $f_1,\ldots,f_k$ and $C_1,\ldots,C_k$ which track the k best maps seen so far and their costs. Use C_k instead of $C_{\min}$ to prune the search space, and only prune when the cost bound is greater than or equal to C_k. When the search is completed, $f_1,\ldots,f_k$ are the k maps with the lowest cost.

One way to drastically speed up the search in inexact ASI and top-k inexact ASM is to set an initial value for U. However, this approach can lead to no solutions being found if the chosen value is lower than the cost of the optimal solution $C(f^\star;\mathcal{G}_t,\mathcal{G}_w)$.

III. EXPERIMENTS

We test our algorithms on the AIDA Version 2.1.2 dataset created by Pacific Northwest National Laboratory (PNNL) for the DARPA-MAA program. This dataset consists of a knowledge graph collected from news articles about political events in the Ukraine. Provided also is a measure of distance between attributes, which can be used to construct the corresponding graph edit distance. The world graph has 98,817 nodes and 138,127 edges. Three templates are provided, each with six different variations labeled A-F. These templates are much smaller than the world graph, consisting of 11-33 nodes and 11-40 edges. These templates were created by taking a known "ground truth" existing subgraph and adding increasing levels of noise. The A template has no noise and is guaranteed to have at least one exact match. Provided also is the ground truth mapping for the A version of each template. All experiments were run on an HP Z8 G4 Workstation with two 12-core 6136 3.0 2666MHz CPUs.

We perform top-k inexact ASM on these templates, with $k = 5$. We use two approaches: the first approach (labeled "Normal") in the table, simply runs the algorithm with no initial cost threshold, while the second uses the ground truth cost bound (labeled "GTCB") from the A template to derive an appropriate initial cost threshold for the other templates in each series (1,2,3). To identify this threshold, we first impose the matching from the ground truth, then set our algorithm to find optimal assignments for any remaining nodes that were not included in the ground truth. Using only this initial cost threshold, we discard all other matching information from the ground truth and proceed to optimize over the space of matches with lower cost than the cost of the ground truth. The cost of the ground truth for each template is listed in the Ground Truth column under Cost. In real world contexts, the ground truth is unknown. However, it is not unreasonable that another, possibly suboptimal, approximate match exists that we can use for the purpose of setting the initial cost threshold to restricting the solution space.

The results of the two approaches on the AIDA Version 2.1.2 dataset are shown in Table I. Although we perform top-k matching, we list only one cost for each template; this is because for all templates considered, all 5 matches found were of the same cost. We observe that for all of the A templates, both the normal and GTCB approaches converge to the expected result of an exact match with 0 graph edit distance. For all templates other than 1C-F, both approaches were able to complete within the runtime limit of 46.5 hours, and arrive at the optimal solutions. For 1C-F, the normal version was aborted due to the runtime limit, resulting in the best solutions found during that time limit. For the GTCB version, only template 1E was aborted early, after 35.9 hours. This was done because, when examining the branching structure of the search algorithm, the approach was deemed unlikely to complete. The other three templates completed and reached optimal solutions.

After identifying optimal solutions (for all templates except 1E), we then apply the approach for inexact ASM discussed in Section II-D to find all optimal solutions by setting the cost threshold to the cost of an optimal solution. The number of optimal solutions found is listed in Table II. In the case of 1E, we set this cost threshold to be the cost of the best solution found; we observe that all found solutions were of the same cost.

For templates 1D, 1E, and 1F, the optimal solution search was cut off after 209 hours due to time and memory limitations. We believe that there exist more solutions than those that were found, possibly orders of magnitude more. We have marked these templates with a $>$ in Table II to indicate this.

A. Analysis of Solution Space

For real world applications it is important to understand the entire solution space rather than finding just one match. Especially for security applications, the match may point to specific people or places, some of which could be imposters. We show an example here of a map from the template to the full solution space - something that could be visualized for analysts trying to understand the connectivity of the template nodes in the full solution space. After using the methods discussed in prior sections to discover subgraph matchings of minimal cost, we can appeal to symmetries apparent in the template graph and world graph to attain a more compact representation of the solution space. Symmetry in graph structures has been studied in depth in the context of subgraph matching [15], [31], [16], [32]. It is a confounding factor for subgraph discovery that can lead to redundant work while exploring symmetric areas of the graph. The kind of symmetry most often utilized is structural equivalence of nodes, i.e., nodes are equivalent if they have the same label and the same neighborhood structure (i.e., the same neighbors and same labels on edges connected to those neighbors, see Figure 1).

In the matching problems on the AIDA knowledge graph, symmetric structures in the graph account for the combinatorial explosion in solutions for certain template graphs, especially in instances where more noise is

TABLE I

RESULTS FOR THE AIDA VERSION 2.1.2 DATASET, SHOWING TIME TAKEN AND THE COST OF THE BEST MATCH THAT WAS FOUND. THE ALGORITHM WAS CUT OFF IF IT FAILED TO COMPLETE WITHIN 46.5 HOURS, TAKING THE BEST MATCH THAT IT HAD FOUND SO FAR.

	Time		Cost		
Template	Normal	GTCB	Normal	Ground Truth	GTCB
1A	27.8 min	1.5 sec	0.0	0.0	0.0
1B	24.5 min	0.2 sec	0.347	0.351	0.347
1C	46.5 hrs	5.48 hrs	8.783	3.610	2.254
1D	46.5 hrs	64.1 min	15.470	1.639	1.636
1E	46.5 hrs	35.9 hrs	16.194	2.642	2.638
1F	46.5 hrs	76.7 min	13.596	2.328	1.772
2A	2.25 min	0.06 sec	0.0	0.0	0.0
2B	3.07 min	0.15 sec	0.307	0.335	0.307
2C	6.35 min	10.6 sec	4.070	4.891	4.070
2D	12.4 min	32.9 sec	3.839	5.942	3.839
2E	4.58 min	22.4 sec	3.848	4.533	3.848
2F	18.5 min	42.3 sec	4.012	4.704	4.012
3A	2.33 min	0.08 sec	0.0	0.0	0.0
3B	2.27 sec	0.15 sec	0.145	0.191	0.145
3C	4.10 min	3.56 sec	2.452	2.452	2.452
3D	10.3 min	2.48 min	2.312	2.585	2.312
3E	20.8 min	10.9 min	2.472	3.471	2.472
3F	4.80 min	0.94 sec	1.314	1.348	1.314

present. Figure 2 demonstrates various forms of symmetry in the world graph and how they affect the structure of the solution space for Template 1B. We present the template graph and its matches to a subgraph of the world graph containing only those nodes appearing in an optimal solution. We color each node in the template graph the same as its candidates in the world graph. The blue and orange nodes in the world graph are two groups of structurally equivalent nodes and so can be swapped out arbitrarily in any solution and maintain a matching of the same cost. The red and lavender groups of world nodes present a slightly more complex form of symmetry (automorphic). Exploiting it for purposes of generating more solutions would require a more intelligent scheme that assigns the red and lavender template nodes as a unit. We leave automorphic symmetry for future work and only consider structural symmetry in this paper. In this example, the problem has 6120 optimal cost solutions; applying structural symmetry, we can reduce it to 2520 solutions from which we can generate the rest.

An a posteriori analysis of the solutions generated by our search enables us to compress the solution space as well as to illustrate potential ways to significantly speed up the subgraph search. We take the subgraph of world nodes that appear in a minimal cost solution and compute groups of nodes that are structurally equivalent. Then to compress the solution space, we eliminate any solution that can be generated by swapping out equivalent world nodes in another mapping. We do this by examining the classes of solutions generated by interchanging equivalent nodes and identifying a representative from each.

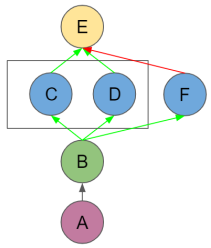

Fig. 1. Structural Equivalence: Nodes C and D are structurally equivalent and F is equivalent to neither. C, D, and F have the same node label and same set of neighbors. However, the edge connecting F to E has a different label than the edges connecting C and D to E, so it F is not equivalent to C or D.

Table II demonstrates the extent to which structural symmetry reduces the description of the solution space to a smaller set. We also list the number of world nodes that appear in a minimal cost solution as well as the number of structural equivalence classes to show the level of equivalence in the solution space. If the number of equivalence classes is significantly smaller than the number of world nodes, there is a great deal of equivalence. We observe that in certain cases, e.g., templates 1E and 1F, the size of the representative solution set is nearly ten

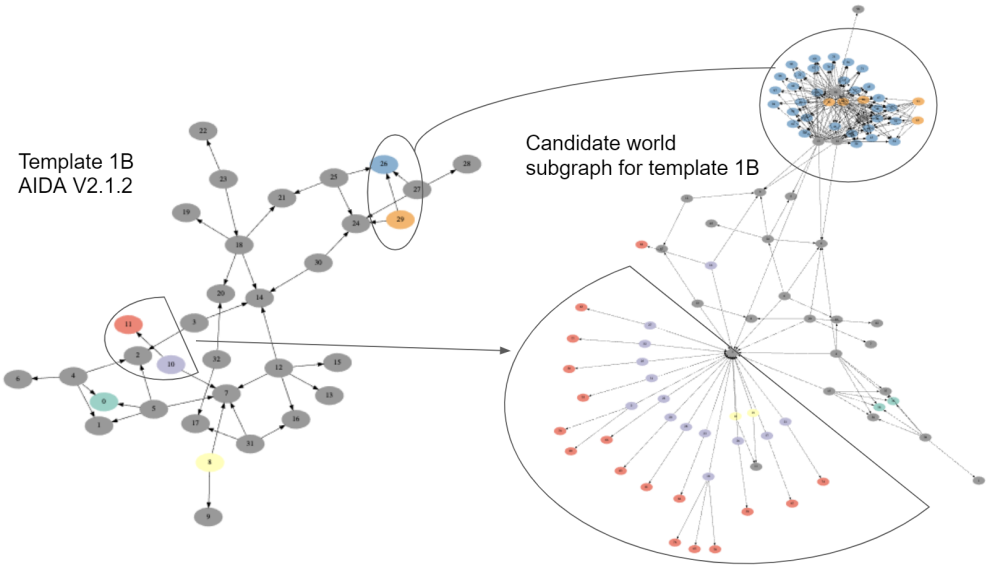

Fig. 2. Template graph and world subgraph for Problem 1B. Colored groups of nodes in the world graph are candidates for nodes of the same color in the template graph. The long arrows and shapes denote corresponding groupings.

TABLE II
THE NUMBER OF SOLUTIONS, REPRESENTATIVE SOLUTIONS, CANDIDATE WORLD NODES, AND EQUIVALENCE CLASSES FOR EACH SUBGRAPH MATCHING PROBLEM FROM THE AIDA VERSION 2.1.2 DATASET. FOR TEMPLATES 1D-F, THE CODE WAS TERMINATED DUE TO RUNTIME CONSTRAINTS BEFORE ALL SOLUTIONS COULD BE FOUND.

	Template					
	1A	1B	1C	1D	1E	1F
# Solutions	6	6120	324	>392k	>382k	>400k
# Rep. Sols.	3	2520	162	>315k	>34k	>58k
# World Nodes	38	99	47	109	3169	3141
# Eq. Classes	37	95	46	105	1686	1692
	2A	2B	2C	2D	2E	2F
# Solutions	78	1400	39	13780	1248	17368
# Rep. Sols.	6	60	3	1160	128	4160
# World Nodes	29	41	27	45	37	48
# Eq. Classes	17	23	15	30	26	36
	3A	3B	3C	3D	3E	
# Solutions	6	6	36	198	198	
# Rep. Sols.	4	4	16	92	92	
# World Nodes	16	16	21	26	23	
# Eq. Classes	15	15	18	23	23	

times smaller. If an algorithm were to efficiently compute these symmetries and incorporate them in a subgraph search, then we might expect speedups of an order of magnitude for these problems.

An analysis through a symmetry lens also exposes how introducing noise into a subgraph matching problem impacts the solution space. Broadly, if more information is known about the labels and neighbors of vertices in both the template and world, there will be less symmetry apparent in the matching problem. This can be seen in Table II with increased noise as we go from A to F which significantly expands the solution space. Intuitively, having more label information allows nodes to distinguish themselves from each other and break symmetry. If we introduce noise into a problem, say by removing a template node's label, then effectively the labels of the world graph become irrelevant as the label cost will be the same.

The graphs provided in the AIDA datasets have three different labels: "rdf:type", which indicates the semantic type of a node (e.g. Person, Location, Vehicle, etc.), "hasName", which gives the names of entities, and "textValue", which contains miscellaneous text information associated with the node. Table III lists the number of equivalence classes for the first set of templates when considering only the "rdf:type" label as compared to considering all labels. As can be seen, when considering only the type label, we have significantly fewer equivalence classes. This is especially apparent in template 1E and 1F for which the minimal cost solution requires two edge mismatches leaving an isolated template node that may match any world graph node as long as the label matches. Of course, when constructing a match-

TABLE III
EQUIVALENCE CLASSES WHEN CONSIDERING ONLY PART OF THE LABEL AND THE FULL LABEL

Template	1A	1B	1C	1D	1E	1F
# Type Eq. Classes	37	67	45	73	36	67
# Full Eq. Classes	37	95	46	105	1686	1692

ing, we must consider all label information; however, understanding the relationship between different levels of noise and symmetry appears critical to fully understanding the solution space of a given problem.

IV. CONCLUSION

We provide an approach for inexact subgraph matching on attributed graphs via optimization of the graph edit distance. Using an approach analogous to constraint propagation, we develop lower bounds on the cost of individual assignments, then combine them using linear sum assignment. Using these cost bounds as a heuristic, we perform a guided depth first search for optimal solutions. We apply our approach to the AIDA V2.1.2 knowledge graph dataset.

In the future, we hope to extend our method to more general types of graph templates, such as templates with pairwise relative attribute constraints. For example, one could impose a constraint that requires two nodes have attribute values whose difference lies within a minimum and maximum range. This is important in the context of temporal data, where two events may be required to occur within a certain time window of each other. Similarly, for spatial data, it may be useful to require two nodes to be within a certain distance of each other.

ACKNOWLEDGMENTS

This material is based on research sponsored by the Air Force Research Laboratory and DARPA under agreement number FA8750-18-2-0066. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Air Force Research Laboratory and DARPA or the U.S. Government.

We thank George Chin, Joseph Cottam, Natalie Heller, Patrick Mackey, Sumit Purohit, and the rest of the team at PNNL for providing us with the AIDA dataset and the graph edit distance metric. We additionally thank Yurun Ge and Xie He for helpful comments and discussions.

APPENDIX A
VARYING EDGEWISE WEIGHTS

In some situations, the costs of certain nodes become more important than other nodes. For example, when a node has a known assignment, it is less important to know the cost associated with that node, and more important to know the costs of its surrounding nodes, so that they may too be assigned candidates using the best possible heuristic.

We repeat the cost bound formulation from Equation (3), but replace $\frac{1}{2}$ by a variable parameter $\alpha(t, t_o)$ which has the property that $\alpha(t, t_o) + \alpha(t_o, t) = 1$ under the constraint $0 \leq \alpha(t, t_o) \leq 1 \forall t, t_o$. If we wish to assign t more importance than t_o, we use $\alpha(t, t_o) = 1, \alpha(t_o, t) = 0$.

$$L_\alpha(t, w; \mathcal{G}_t, \mathcal{G}_w, f) = D_\mathcal{V}(t, w) + \sum_{t_o \in \mathcal{V}_t} \alpha(t, t_o) \Big(D_\mathcal{E}\big((t, t_o), (w, f(t_o))\big) + D_\mathcal{E}\big((t_o, t), (f(t_o), w)\big) \Big).$$

Similarly, we extend $B(t, w; \mathcal{G}_t, \mathcal{G}_w)$ to $B_\alpha(t, w; \mathcal{G}_t, \mathcal{G}_w)$ defined as follows

$$B_\alpha(t, w; \mathcal{G}_t, \mathcal{G}_w) := D_\mathcal{V}(t, w) + \sum_{t_o \in \mathcal{V}_t} \alpha(t, t_o) \min_{w_o \in \mathcal{V}_w} \Big(D_\mathcal{E}\big((t, t_o), (w, w_o)\big) + D_\mathcal{E}\big((t_o, t), (w_o, w)\big) \Big).$$

In practice, this is used during the search algorithm to put less weight on assigned nodes. When a node is given an assignment, it is also given a relative weight of $\alpha = 0$, while its neighbors are given a weight of $\alpha = 1$ with respect to that node. For an edge between two assigned nodes, we default to $\alpha = \frac{1}{2}$.

APPENDIX B
RESTRICTING CANDIDATES DURING MINIMIZATION

Within the context of the search approach detailed in Section II-D, at certain points in the algorithm, we have an upper bound U on the cost. At this point in the algorithm, the exact value of the cost bound is not needed if it is greater than or equal to U (or strictly greater in the context of inexact ASM).

Thus, we can instead define the local cost bound as

$$B(t, w; \mathcal{G}_t, \mathcal{G}_w) := \min \Big(U, D_\mathcal{V}(t, w) + \sum_{t_o \in \mathcal{V}_t} \alpha(t, t_o) \min_{w_o \in C(t_o)} \Big[D_\mathcal{E}\big((t, t_o), (w, w_o)\big) + D_\mathcal{E}\big((t_o, t), (w_o, w)\big) \Big] \Big)$$

where $C(t_o)$ is defined to be the set of world nodes w_o for which the known constrained cost bound $G(t_o, w; \mathcal{G}_t, \mathcal{G}_w)$ is strictly less than U (less than or equal to for inexact ASM). Doing so drastically improves computational performance and provides a tighter bound on costs which lie below U. This also refines cost bounds analogously to constraint propagation; as tighter global cost bounds G are found, this leads to tighter local cost bounds, which are then used to compute even tighter global cost bounds until we reach a final set of bounds.

REFERENCES

[1] D. Conte, P. Foggia, C. Sansone, and M. Vento, "Graph matching applications in pattern recognition and image processing," in *Proceedings 2003 International Conference on Image Processing (Cat. No.03CH37429)*, vol. 2, Sep. 2003, pp. II–21.

[2] V. Carletti, P. Foggia, A. Saggese, and M. Vento, "Introducing VF3: A new algorithm for subgraph isomorphism," *Graph-Based Representations in Pattern Recognition*, pp. 128–139, 2017.

[3] V. Carletti, P. Foggia, A. Saggese, and M. Vento, "Challenging the time complexity of exact subgraph isomorphism for huge and dense graphs with vf3," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 40, no. 4, pp. 804–818, April 2018.

[4] J. R. Ullmann, "An algorithm for subgraph isomorphism," *J. ACM*, vol. 23, no. 1, pp. 31–42, Jan. 1976.

[5] L. P. Cordella, P. Foggia, C. Sansone, and M. Vento, "A (sub)graph isomorphism algorithm for matching large graphs," *IEEE Trans. on Pattern Analysis and Machine Intell.*, vol. 26, no. 10, pp. 1367–1372, Oct 2004.

[6] V. Bonnici, R. Giugno, A. Pulvirenti, D. Shasha, and A. Ferro, "A subgraph isomorphism algorithm and its application to biochemical data," *BMC bioinformatics*, vol. 14, no. 7, p. S13, 2013.

[7] J. J. McGregor, "Relational consistency algorithms and their application in finding subgraph and graph isomorphisms," *Information Sciences*, vol. 19, no. 3, pp. 229–250, 1979.

[8] J. Larrosa and G. Valiente, "Constraint satisfaction algorithms for graph pattern matching," *Mathematical Structures in Computer Science*, vol. 12, no. 4, p. 403–422, 2002.

[9] S. Zampelli, Y. Deville, and C. Solnon, "Solving subgraph isomorphism problems with constraint programming," *Constraints*, vol. 15, pp. 327–353, 07 2010.

[10] C. Solnon, "AllDifferent-based Filtering for Subgraph Isomorphism," *Artificial Intell.*, vol. 174, pp. 850–864, Aug. 2010.

[11] C. McCreesh and P. Prosser, "A parallel, backjumping subgraph isomorphism algorithm using supplemental graphs," in *Int. Conf. on Principles and Practice of Constraint Programming*, G. Pesant, Ed. Springer Int. Publishing, 2015, pp. 295–312.

[12] J. D. Moorman, Q. Chen, T. K. Tu, Z. M. Boyd, and A. L. Bertozzi, "Filtering methods for subgraph matching on multiplex networks," in *2018 IEEE International Conference on Big Data (Big Data)*, Dec 2018, pp. 3980–3985.

[13] H. He and A. Singh, "Graphs-at-a-time: query language and access methods for graph databases," in *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*. ACM, 2008, pp. 405–418.

[14] P. Zhao and J. Han, "On graph query optimization in large networks," *Proceedings of the VLDB Endowment*, vol. 3, no. 1-2, pp. 340–351, 2010.

[15] W. Han, J. Lee, and J. Lee, "Turbo iso: towards ultrafast and robust subgraph isomorphism search in large graph databases," in *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*. ACM, 2013, pp. 337–348.

[16] F. Bi, L. Chang, X. Lin, L. Qin, and W. Zhang, "Efficient subgraph matching by postponing cartesian products," in *Proceedings of the 2016 International Conference on Management of Data*. ACM, 2016, pp. 1199–1214.

[17] V. Ingalalli, D. Ienco, and P. Poncelet, "Sumgra: Querying multigraphs via efficient indexing," in *International Conference on Database and Expert Systems Applications*. Springer, 2016, pp. 387–401.

[18] B. Bhattarai, H. Liu, and H. H. Huang, "Ceci: Compact embedding cluster index for scalable subgraph matching," in *Proceedings of the 2019 International Conference on Management of Data*, 2019, pp. 1447–1462.

[19] R. Pienta, A. Tamersoy, H. Tong, and D. H. Chau, "Mage: Matching approximate patterns in richly-attributed graphs," in *2014 IEEE International Conference on Big Data (Big Data)*. IEEE, 2014, pp. 585–590.

[20] H. Jin, X. He, Y. Wang, H. Li, and A. L. Bertozzi, "Noisy subgraph isomorphisms on multiplex networks," in *2019 IEEE International Conference on Big Data (Big Data)*. IEEE, 2019, pp. 4899–4905.

[21] A. Kopylov and J. Xu, "Filtering strategies for inexact subgraph matching on noisy multiplex networks," in *2019 IEEE International Conference on Big Data (Big Data)*, 2019, pp. 4906–4912.

[22] Y. Liang and P. Zhao, "Similarity search in graph databases: A multi-layered indexing approach," in *2017 IEEE 33rd International Conference on Data Engineering (ICDE)*. IEEE, 2017, pp. 783–794.

[23] D. Sussman, Y. Park, C. E. Priebe, and V. Lyzinski, "Matched filters for noisy induced subgraph detection," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pp. 1–1, 2019.

[24] A. Sanfeliu and K. Fu, "A distance measure between attributed relational graphs for pattern recognition," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. SMC-13, no. 3, pp. 353–362, 1983.

[25] X. Gao, B. Xiao, D. Tao, and X. Li, "A survey of graph edit distance," *Pattern Analysis and applications*, vol. 13, no. 1, pp. 113–129, 2010.

[26] H. W. Kuhn, "The hungarian method for the assignment problem," *Naval Research Logistics Quarterly*, vol. 2, no. 1-2, pp. 83–97, 1955.

[27] J. Munkres, "Algorithms for the assignment and transportation problems," *Journal of the Society for Industrial and Applied Mathematics*, vol. 5, no. 1, pp. 32–38, 1957.

[28] R. Jonker and A. Volgenant, "A shortest augmenting path algorithm for dense and sparse linear assignment problems," *Computing*, vol. 38, no. 4, pp. 325–340, 1987.

[29] D. P. Bertsekas, "The auction algorithm: A distributed relaxation method for the assignment problem," *Annals of Operations Research*, vol. 14, no. 1, pp. 105–123, 1988.

[30] D. F. Crouse, "On implementing 2D rectangular assignment algorithms," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 52, no. 4, pp. 1679–1696, 2016.

[31] X. Ren and J. Wang, "Exploiting vertex relationships in speeding up subgraph isomorphism over large graphs," *Proceedings of the VLDB Endowment*, vol. 8, no. 5, pp. 617–628, 2015.

[32] T. Nguyen, D. Yang, Y. Ge, H. Li, and A. L. Bertozzi, "Applications of structural equivalence to subgraph isomorphism on multichannel multigraphs," in *2019 IEEE International Conference on Big Data (Big Data)*. IEEE, 2019, pp. 4913–4920.