Schema-Guided Event Graph Completion

Hongwei Wang[1], Zixuan Zhang[1], Sha Li[1], Jiawei Han[1],
Yizhou Sun[2], Hanghang Tong[1], Joseph P. Olive[3], Heng Ji[1]
[1]University of Illinois Urbana-Champaign [2]University of California, Los Angeles
[3]JP Olive S&T Management LLC
{hongweiw, zixuan11, sha12, hanj, htong, hengji}@illinois.edu, yzsun@cs.ucla.edu, joseph.olive.ctr@darpa.mil

ABSTRACT

We tackle a new task, *event graph completion*, which aims to predict missing event nodes for event graphs. Existing link prediction or graph completion methods have difficulty dealing with event graphs, because they are usually designed for a single large graph such as a social network or a knowledge graph, rather than multiple small dynamic event graphs. Moreover, they can only predict missing edges rather than missing nodes. In this work, we propose to utilize *event schema*, a template that describes the stereotypical structure of event graphs, to address the above issues. Our schema-guided event graph completion approach first maps an instance event graph to a subgraph of the schema graph by a heuristic subgraph matching algorithm. Then it predicts whether a candidate event node in the schema graph should be added into the instantiated schema subgraph by characterizing two types of local topology of the schema graph: *neighbors* of the candidate node and the subgraph, and *paths* that connect the candidate node and the subgraph. These two modules are later combined together for the final prediction. We also propose a self-supervised strategy to construct training samples, as well as an inference algorithm that is specifically designed to complete event graphs. Extensive experimental results on four datasets demonstrate that our proposed method achieves state-of-the-art performance, with 4.3% to 19.4% absolute F1 gains over the best baseline method on the four datasets.

1 INTRODUCTION

Event graphs [6] are structured representation of real-world complex events that are automatically extracted from natural language texts such as news. An event graph consists of event nodes and entity nodes, as well as their relations including event-event temporal links, event-entity argument links, and entity-entity relations. The upper part of Figure 1 gives an example of a complex bombing event extracted from a news report, where yellow and green nodes denote event mentions and entity mentions, respectively.

Nonetheless, a practical issue on event graphs is that they are often incomplete and noisy, because existing information extraction methods have limited performance to understand ambiguous natural language, and many atomic events are often not explicitly described in the source documents such as news articles. For example, in the instance graph in Figure 1, the INJURE:0 event probably has an argument PLACE linked to entity THE SQUARE, and the next-step event after IDENTIFY:0 is also missing. A common solution to this problem is to utilize off-the-shelf algorithms for link prediction [14, 26, 34] or graph completion [7, 25, 35] to refine raw event graphs. However, they have difficulty dealing with event graphs in the following two aspects: (1) Existing link prediction or graph completion methods usually focus on a *single large* graph (e.g., an online social network or knowledge graph with thousands

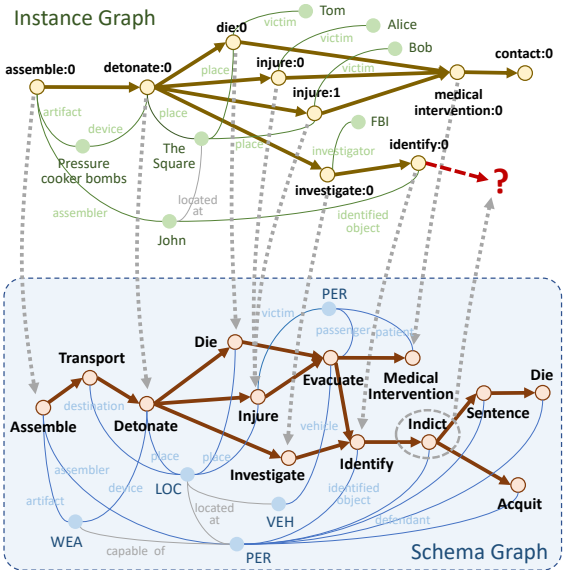


Figure 1: An illustrative example of schema-guided event graph completion. The above is an (incomplete) instance graph about a bombing event, and the below is the schema of general improvised explosive device (IED) bombing. The grey dotted arrows between the two graphs denote the event node matching relation (entity node matching is omitted for clarity). The schema can inject human knowledge into instance graphs and help predict missing events such as INDICT.

or even millions of nodes), but an event graph dataset consists of *multiple small* instance event graphs, each of which is extracted from a cluster of topically-related news articles and contains dozens of nodes only. The instance event graphs are usually independent (e.g., describing different bombing events) but follow the similar pattern (e.g., all bombing events are similar), which, unfortunately, cannot be characterized by existing methods. (2) Existing link prediction or graph completion methods can only detect a missing link between two nodes that already exist in the graph, but they cannot tell if a new node is missing from the graph and how this new node should be connected to existing nodes. For example, in the instance graph in Figure 1, it is unlikely to predict the next-step event after IDENTIFY:0 with the existing link prediction or graph completion methods. However, note that a significant difference between event graphs and knowledge graphs is that, knowledge graphs are entity-centric and consist of static entity-entity relations, while event graphs are event-centric and dynamic, where new events and entities may appear as the graph evolves. Therefore,

predicting missing events is a realistic, informative, and exciting task for event graph completion [15].

To address the limitations of existing methods when applied to event graphs, a promising solution is to make use of event schemas. An *event schema* (a.k.a. complex event template) is a generic and abstract representation of a specific type of complex events that encodes their stereotpyical structure. Event schemas can be either generated automatically by machines [8, 15, 16, 28, 29] from a large collection of historical event graphs, or manually curated by human. The lower part of Figure 1 illustrates the schema of general improvised explosive device (IED) bombing, where red and blue nodes represent the types of events and entities, respectively. We propose to utilize event schemas to help solve the two aforementioned issues: An event schema serves as a template for a particular type of complex events, which represents the generalized knowledge in this particular scenario and enables us to model the common pattern of instance graphs; Moreover, an event schema can be seen as a pool of inter-connected candidate events, which provides us with new event nodes that can be added into an incomplete instance event graph (e.g., the INDICT node in Figure 1).

In this paper, we propose a schema-guided approach for event graph completion. Given an incomplete instance event graph, we aim to predict whether a candidate event node from the schema graph is missing for the instance graph. To build the connection between the schema graph and instance graphs, we propose first using a two-stage heuristic subgraph matching algorithm to map an instance graph to a subgraph of the schema graph (see Figure 1 for example), which can greatly reduce the time overhead of the exact subgraph matching algorithm. After the subgraph matching step, our problem is equivalent to inferring whether the candidate node is missing for the matched subgraph of the schema. We therefore explore two types of local topology for a subgraph-node pair within the schema graph: (1) *Neighbors*. It is important to capture the neighboring node types of a given node/subgraph in the schema graph, because they provide us with valuable information about what the nature of the given node/subgraph is. For example, there are two events with the same type of DIE in the schema graph in Figure 1, but the neighboring events of the first DIE (e.g., DETONATE, EVACUATE, MEDICAL INTERVENTION) indicate that its subject is a victim of the IED bombing, while the neighboring events of the second DIE (e.g., SENTENCE, INDICT) indicate that it refers to the death of the attacker. We apply a multi-layer graph neural network (GNN) to aggregate information from multi-hop neighboring nodes in the schema graph, and compute the correlation between the subgraph and the node in terms of their GNN representations. (2) *Paths*. Note that modeling only neighboring node types is not able to identify the distance between the subgraph and the candidate node. It is also important to capture the set of paths connecting them, which can reveal the nature of their relation and enable the model to capture the distance between them. For example, in the schema graph in Figure 1, it is clear that DETONATE is more closely related to the first DIE than the second DIE, since DETONATE and the first DIE are directly connected, while the paths connecting DETONATE and the second DIE are much longer (e.g., DETONATE-INVESTIGATE-IDENTIFY-INDICT-SENTENCE-DIE). We collect all paths connecting the candidate node and the subgraph, then use those paths to calculate their correlation. Finally, we combine these two modules together to predict the probability that the candidate node is missing for the subgraph.

To solve the problem of lacking ground-truth missing events, we propose a self-supervised approach to construct training samples, which masks out a node from a given subgraph and treats the masked node as a missing event to be predicted. We also propose an inference algorithm specifically designed for event graph completion, which expands an incomplete instance graph over the schema by utilizing the structure of the schema graph.

We conduct extensive experiments on four event graph datasets in the scenarios of IED bombing and disease outbreak. Experimental results demonstrate that our method achieves state-of-the-art performance in missing event prediction. For example, the F1 score of our method surpasses the best baseline method by 7.0%, 19.4%, 11.5%, and 4.3% respectively, on the four datasets. Moreover, our ablation study and case study verify the effectiveness of the proposed neighbor module and path module.

2 PROBLEM FORMULATION

We formulate the problem of schema-guided event graph completion as follows. Suppose we have a set of instance event graphs $\mathcal{I} = \{I_1, I_2, \cdots\}$, which are constructed from a set of articles with the same topic (e.g., car bombing). Each instance graph I describes a complex event and consists of an event node set $\{e_i\}$ and an entity node set $\{v_i\}$, in which each node e_i or v_i is instantiated as a real event or entity. In addition, each event or entity is also associated with a specific event or entity type, and we use $\tau(\cdot)$ to denote the mapping function from a node to its type. Accordingly, there are three types of links in instance graphs: (1) event-event temporal link $\langle e_i, e_j \rangle$, which indicates that event e_j happens chronologically after event e_i (e.g., ⟨DETONATE, INJURE⟩); (2) event-entity link $\langle e_i, a, v_j \rangle$, which indicates that event e_i has an argument role a, whose value is entity v_j (e.g., ⟨DETONATE, VICTIM, ALICE⟩); (3) entity-entity link $\langle v_i, r, v_j \rangle$, which indicates that there is a relation r between entity e_i and e_j (e.g., ⟨JOHN, LOCATED AT, THE SQUARE⟩). We also use function $\tau(\cdot)$ to denote the type of a link. Specifically, $\tau(\langle e_i, e_j \rangle) =$ TEMP,[1] $\tau(\langle e_i, a, v_j \rangle) = a$, and $\tau(\langle v_i, r, v_j \rangle) = r$.

Moreover, we also have a schema graph S available, which is a generic representation of instance graphs in $\mathcal{I}$ and characterizes their typical structure. The format of the schema graph S is similar to instance graphs, while the only difference is that nodes in S are not instantiated but are only specified by event or entity types.

Given the training instance graphs $\mathcal{I}$ and the schema graph S, we aim to learn a model that is able to predict missing events for a new and incomplete instance graph. Specifically, for a new instance graph I and a candidate event node $e \in S$, our model aims to predict the probability that e is a missing event for I.[2]

3 OUR APPROACH

Given an incomplete instance event graph as well as the event schema graph, we first perform subgraph matching between them to map the instance graph to a subgraph of the schema graph

[1] We use TEMP to denote the type of a temporal link, and TEMP_REV to denote the type of a reversed temporal link.

[2] It is important to predict not only the type of missing events but also their arguments as well as temporal connections to existing event nodes. We will discuss the details of argument/temporal link completion in Section 3.5.

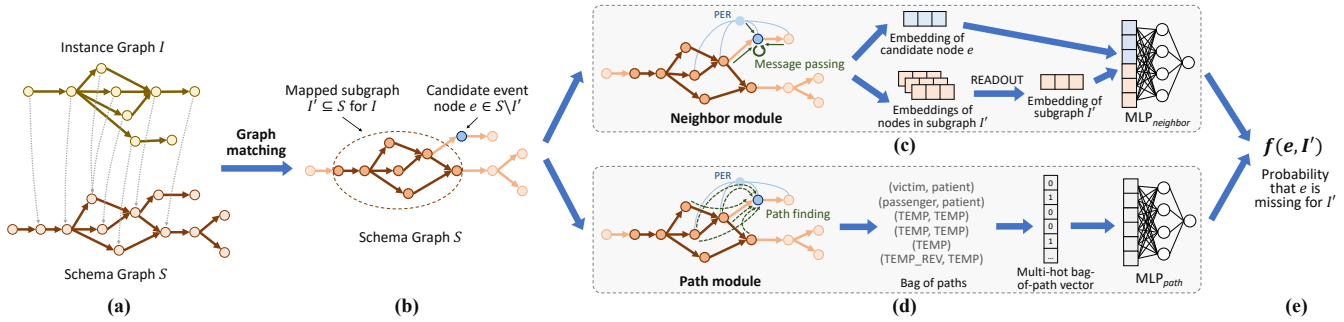


Figure 2: Illustration of the model architecture. Entity nodes are omitted for the sake of clarity. (a) For an instance event graph I and the event schema graph S, a two-stage subgraph matching algorithm is performed to map event nodes in I to event nodes in S. (b) The instance graph I is mapped to a subgraph $I' \subseteq S$. Our goal is to predict whether a candidate event node $e \in S \backslash I'$ is a missing node and thus should be added into I'. (c) The neighbor module. Message passing is performed on the schema graph to learn the representations of all nodes. An entity node PER (person) is drawn here to show that the message passing is performed on the entire schema graph. Embeddings of nodes in I' are aggregated by a READOUT function to compute the embedding of I'. The embeddings of e and I' are fed into $\text{MLP}_{neighbor}$. (d) The path module. All paths connecting e and a node in I' with length no more than a given threshold (2 in this example) are highlighted by green dotted arrows, which forms the bag-of-paths feature for the pair (e, I'). It is then transformed to a multi-hot bag-of-paths vector, which are fed into MLP_{path}. (e) The outputs of $\text{MLP}_{neighbor}$ and MLP_{path} are combined together to compute the final predicting probability $f(e, I')$.

(Section 3.1). Then we design a neighbor module (Section 3.2) and a path module (Section 3.3) to compute the correlation between the mapped subgraph and a candidate event node in the schema. We also discuss the training (Section 3.4) and inference (Section 3.5) procedures of the proposed model. The architecture of our proposed model is shown in Figure 2.

3.1 Subgraph Matching

Since our goal is to predict whether a candidate event node e from the schema graph S is a missing node for an instance graph I, the first step is therefore to perform subgraph matching between S and I. In this way, I can be mapped to a subgraph of S, which enables us to compute the correlation between the instance graph I and the candidate event node e within the scope of schema graph S.

Integer Programming. We can use a binary assignment matrix $\mathbf{X} \in \{0,1\}^{|I|\times|S|}$ to represent the matching solution, in which $\mathbf{X}_{ij} = 1$ if and only if node $i \in I$ matches node $j \in S$. We also use $s_N(i,j)$ to denote the pairwise node similarity for $i \in I$ and $j \in S$, and $s_L(\langle i,j\rangle, \langle k,l\rangle)$ to denote the pairwise link similarity function for $\langle i,j\rangle \in I \times I$ and $\langle k,l\rangle \in S \times S$. Then the subgraph matching problem can be formulated as finding the assignment matrix that maximizes the total matching score:

$$\begin{aligned} &\max_{\mathbf{X}} \sum_{\mathbf{X}_{ij}=1} s_N(i,j) + \sum_{\substack{\mathbf{X}_{ik}=1 \\ \mathbf{X}_{jl}=1}} s_L(\langle i,j\rangle, \langle k,l\rangle) \\ &s.t. \begin{cases} \mathbf{X} \in \{0,1\}^{|I|\times|S|} \\ \sum_{j=1}^{|S|} \mathbf{X}_{ij} \le 1, \text{ for } i = 1, \cdots, |I|, \end{cases} \end{aligned} \tag{1}$$

where the constraints define a many-to-one node mapping: a node in I can be mapped to at most one node in S, while a node in S can be mapped to multiple nodes in I. This is because an instance graph may have repeated event types (e.g., two Injure events in Figure 1), which should be mapped to the same event node in the schema.

It is worth noting that Eq. (1) defines a 0-1 integer programming problem [2], which is NP-complete. To improve the time efficiency of subgraph matching, we notice the following two facts: (1) The instance graph and the schema graph are both event-node-centric. Therefore, we can focus on matching event nodes in the two graphs, which will greatly reduce the entire searching space. Once event nodes are matched, entity nodes can be naturally matched based on their event argument roles. (2) Event types play a decisive role when determining whether two event nodes are matched. For example, in Figure 1, the die:0 node in the instance graph should be matched to Die in the schema graph rather than Injure since they have the same event type, even if the arguments of die:0 are different from Die but are exactly the same as Injure.

A Two-Stage Heuristic Subgraph Matching Scheme. Taking into consideration the above two facts, we propose a two-stage heuristic subgraph matching scheme between instance and schema graphs. For an event node $e_i \in I$ to be matched, we first identify the event node(s) in S whose type is the same as e_i:[3]

$$E_i = \{e_j \in S \mid \tau(e_j) = \tau(e_i)\}, \tag{2}$$

Based on the size of E_i, we have the following three cases:

- $|E_i| = 0$. This means that e_i will not be matched to any event node in S (e.g., the contact:0 node in Figure 1).
- $|E_i| = 1$. In this case, we end up with a unique node in S, which is taken as the matching node for e_i (e.g., the assemble:0 node in Figure 1).

[3] If event types in instance graphs and the schema do not follow the same naming style and thus cannot be compared directly, we can use embedding-based fuzzy matching: $e^* = \arg\max_{e_j \in S} \text{LM}(\tau(e_i))^\top \text{LM}(\tau(e_j))$, where $\text{LM}(\cdot)$ is a language model encoder such as BERT [4].

- $|E_i| > 1$, i.e., there are more than one event node in S that can be matched to e_i, since there may be multiple events in S that have the same event type. For example, the DIE:0 node in Figure 1 can be matched to two DIE nodes.

For the last case above, we design an additional stage for event matching, which is based on the similarity of neighbor node types. Specifically, for each event $e_j \in E_i$, we identify the set of types of e_j's one-hop previous events $\mathcal{P}$, one-hop following events $\mathcal{F}$, and argument roles $\mathcal{A}$, in schema graph S:

$$\begin{aligned} \mathcal{P}(e_j; S) &= \{\tau(e_k) \mid \langle e_k, e_j \rangle \in S\}, \\ \mathcal{F}(e_j; S) &= \{\tau(e_k) \mid \langle e_j, e_k \rangle \in S\}, \\ \mathcal{A}(e_j; S) &= \{a \mid \langle e_j, a, v_k \rangle \in S\}. \end{aligned} \tag{3}$$

We can also identify the above three sets for e_i in instance graph I, i.e., $\mathcal{P}(e_i; I)$, $\mathcal{F}(e_i; I)$, and $\mathcal{A}(e_i; I)$. Then we calculate the Jaccard index[4] J of the three corresponding set pairs, and select the node with the highest total Jaccard index as the matching result:

$$\begin{aligned} e^* = \arg\max_{e_j \in E_i} & J\big(\mathcal{P}(e_j; S), \mathcal{P}(e_i; I)\big) + J\big(\mathcal{F}(e_j; S), \mathcal{F}(e_i; I)\big) \\ & + J\big(\mathcal{A}(e_j; S), \mathcal{A}(e_i; I)\big). \end{aligned} \tag{4}$$

Note that in this two-stage matching scheme, the first stage (Eq. (2)) and the second stage (Eq. (4)) exactly correspond to the two terms in the objective function in Eq. (1), i.e., node similarity and link similarity. In most cases, this two-step matching scheme will return a unique matching node $e^* \in S$ for the input node $e_i \in I$. But if there are still more than one nodes in S that have the highest total Jaccard similarity with e_i, we will randomly select one as the matching result.

Here we analyze the time complexity of our proposed subgraph matching algorithm. We use N_I and N_S to denote the number of event nodes in I and S, respectively, and d to denote their average node degree. For a given event $e_i \in I$, the complexity of calculating E_i in Eq. (2) is $O(N_S)$, and the complexity of calculating $\mathcal{P}(e_i; I)$, $\mathcal{F}(e_i; I)$, and $\mathcal{A}(e_i; I)$ is d. In the worst case where $|E_i| = N_S$, we need to calculate Eq. (3) as well as the Jaccard index for all nodes in S, whose complexity is $O(N_S(d + d)) = O(N_S d)$. Therefore, the total complexity is $O(N_I(N_S + d)N_S d) = O(N_I N_S^2 d)$, which is polynomial w.r.t. the size of I and S.

3.2 Neighbors

After subgraph matching between the instance graph I and the schema graph S, I is mapped to I', which is a subset of event nodes in S. Our goal is therefore to learn a predicting function $f(e, I')$, which outputs the probability of whether a new event $e \in S \backslash I'$ is a missing node for $I' \subseteq S$. In this subsection, we measure the correlation between I' and e in terms of their neighbors.

To learn the representation of nodes as well as subgraphs in the schema, we choose GNNs, which utilize graph structure and node features to learn a representation vector for each node, as our base model. Typical GNNs follow a neighborhood aggregation strategy, which iteratively updates the representation of an atom by aggregating representations of its neighbors. Formally, the k-th layer of a GNN is:

$$\mathbf{h}_i^k = \mathrm{AGG}\left(\left\{\mathbf{h}_j^{k-1}\right\}_{j \in \mathcal{N}(i) \cup \{i\}}\right), \tag{5}$$

where $\mathbf{h}_i^k$ is the representation vector of node $i \in S$ at the k-th layer ($\mathbf{h}_i^0$ is initialized as the one-hot vector of i's node type), and $\mathcal{N}(i)$ is the set of nodes directly connected to i. The choice of AGG function is essential to designing GNNs, and a number of GNN architectures have been proposed [9, 13, 24]. For example, in Graph Convolutional Networks (GCN) [13], AGG is implemented as weighted average with respect to the inverse of square root of node degrees:

$$\mathbf{h}_i^k = \sigma\Big(\mathbf{W}^k \sum\nolimits_{j \in \mathcal{N}(i) \cup \{i\}} \alpha_{ij} \mathbf{h}_j^{k-1} + \mathbf{b}^k\Big), \tag{6}$$

where $\alpha_{ij} = 1/\sqrt{|\mathcal{N}(i)| \cdot |\mathcal{N}(j)|}$, $\mathbf{W}^k$ and $\mathbf{b}^k$ are a learnable matrix and bias, respectively, and σ is an activation function. We use GCN as the implementation of GNN, while the performance of other GNN models is reported in Appendix B.

Suppose the number of GNN layers is K. The final embeddings of event e and events in the subgraph I' are therefore $\mathbf{h}_e^K$ and $\left\{\mathbf{h}_{e_i}^K\right\}_{e_i \in I'}$, respectively. Then, a readout function is used to aggregate event embeddings in I' and output the embedding of I':

$$\mathbf{h}_{I'}^K = \mathrm{READOUT}\left(\left\{\mathbf{h}_{e_i}^K\right\}_{e_i \in I'}\right). \tag{7}$$

The READOUT function can be summation, average, or a more sophisticated attention-based aggregation, since the importance of events in a subgraph may be different with respect to the given event e:

$$\mathbf{h}_{I'}^K = \sum\nolimits_{e_i \in I'} \beta_i \mathbf{h}_{e_i}^K, \tag{8}$$

where $\beta_i = \frac{\mathbf{h}_{e_i}^{K\top} \mathbf{h}_e^K}{\sum_{e_i \in I'} \mathbf{h}_{e_i}^{K\top} \mathbf{h}_e^K}$ is the attention weight. We will report the performance of different READOUT functions in Appendix B.

Finally, the embeddings of e and I' are concatenated, followed by a Multi-Layer Perceptron (MLP) to predict the probability that e is missing for I':

$$p_{neighbor}(e, I') = \mathrm{MLP}_{neighbor}\left(\left[\mathbf{h}_e^K, \mathbf{h}_{I'}^K\right]\right). \tag{9}$$

3.3 Paths

Note that when using GNN to process node neighbors for the mapped subgraph I' and the candidate event node e, we use node types as the initial node feature, which leads to a potential issue that our model is not able to identify the distance between I' and e. For example, think of an extreme case where we have two event nodes e_1 and e_2 in the schema graph, whose neighbor structures within K hops are exactly the same in terms of node types. In this case, the final embeddings of event e_1 and e_2 are also the same if we use a GNN with no more than K layers to process their neighbors. This implies that, for any subgraph $S' \subseteq S$, the predicted probability of $p_{neighbor}(e_1, S')$ and $p_{neighbor}(e_2, S')$ will also be the same, according to Eq. (9). However, this is not always reasonable, since a subgraph S' may be much closer to e_1 than e_2 in the schema graph, thus $p_{neighbor}(e_1, S')$ should be much larger than $p_{neighbor}(e_2, S')$.

To make our model capture the distance information, we model the connectivity pattern between a subgraph and a node, which is characterized by paths connecting them in the schema graph.

[4] The Jaccard index between two sets is defined as $J(A, B) = \frac{|A \cap B|}{|A \cup B|}$. If A and B are both empty, $J(A, B) = 1$.

Specifically, a path connecting two nodes s and t is a sequence of nodes and edges:

$$s \xrightarrow{\langle s,i \rangle} i \xrightarrow{\langle i,j \rangle} j \cdots k \xrightarrow{\langle k,t \rangle} t, \tag{10}$$

where $\langle i, j \rangle$ is the edge connecting node i and j, and each node in the path is unique[5]. In this work, we use the types of edges in a path to represent the path, i.e.,

$$\big(\tau(\langle s,i \rangle), \tau(\langle i,j \rangle), \cdots, \tau(\langle k,t \rangle)\big). \tag{11}$$

We use $\mathcal{P}^{\leq L}_{s \to t}$ to denote the set of all paths connecting s and t with length of no more than L, where L is a given hyperparameter. For example, as shown in Figure 1, there are two paths with length of one or two that connect Transport and Detonate in the schema graph, i.e., $\mathcal{P}^{\leq 2}_{\text{Transport} \to \text{Detonate}} = \{(\text{TEMP}), (\text{destination}, \text{place})\}$.

For an given event e and a subgraph I' in the schema graph, we collect all paths connecting e and every event node in I' as the path set for (e, I'):

$$\mathcal{P}^{\leq L}_{e \to I'} = \bigcup_{e_i \in I'} \mathcal{P}^{\leq L}_{e \to e_i}, \tag{12}$$

which is then transformed into a multi-hot bag-of-paths vector $\mathbf{p}^{\leq L}_{e \to I'}$, where each entry indicates that if a particular path exists in $\mathcal{P}^{\leq L}_{e \to I'}$. We use another MLP to take $\mathbf{p}^{\leq L}_{e \to I'}$ as input and output the probability that e is missing for I':

$$p_{path}(e, I') = \text{MLP}_{path}\left(\mathbf{p}^{\leq L}_{e \to I'}\right). \tag{13}$$

Finally, our predicting function $f(e, I')$ is implemented by combining the output of the neighbor module in Eq. (9) and the path module in Eq. (13):

$$f(e, I') = \frac{1}{2}\left(p_{neighbor}(e, I') + p_{path}(e, I')\right). \tag{14}$$

3.4 Training

A potential issue of training the proposed model is the lack of ground-truth for predicting missing events for a given instance graph. Therefore, we propose a *self-supervised* loss as the training target. Specifically, we first map each instance graph I in the training data to the schema graph S and get the matched subgraph I'. Then for each event node $e \in I'$, we mask e out from I' and try to predict e using the rest of I'. In other words, we treat $(e, I' \backslash e)$ as a positive training sample for each $e \in I'$. Meanwhile, we can randomly sample an event node outside I', i.e., $e \in S \backslash I'$, and treat (e, I') as a negative sample. The total loss function is therefore as follows:

$$L = \frac{1}{|\mathcal{I}|} \sum_{I \in \mathcal{I}} \frac{1}{|S|} \left(\sum_{e \in I'} C\Big(f(e, I' \backslash e), 1\Big) + \sum_{e \in S \backslash I'} C\Big(f(e, I'), 0\Big) \right), \tag{15}$$

where $\mathcal{I}$ is the set of training instance graphs, I' is a subgraph of S that I is mapped to, and C is cross-entropy loss. In Eq. (15), the first term is the loss for positive samples, and the second term is the loss for negative samples.

Note that the training data constructed by Eq. (15) may be unbalanced, i.e., the total numbers of positive and negative samples may be significantly different. In this case, we can use downsampling strategy to re-balance the dataset.

[5]The nodes in a path are required to be unique because a loop does not provide any additional information and thus should be cut off from the path.

Algorithm 1: Inference procedure

Input: An incomplete instance graph I, the schema graph S, the trained model $f(e, I') : S \times 2^S \mapsto [0, 1]$
Output: The completed graph $\hat{I}$

1 $\hat{I} \leftarrow I$;
2 Map I to $I' \subseteq S$ using the two-stage subgraph matching algorithm presented in Section 3.1;
3 Candidate event node set $C \leftarrow S \backslash I'$;
4 **while** *C is not empty* **do**
5 $c' \leftarrow \arg\max_{e \in C:\ dist(e, I') = 1} f(e, I')$;
6 **if** $f(c', I') > Threshold$ **then**
7 $I' \leftarrow I' \cup \{c'\}, C \leftarrow C \backslash c'$;
8 Add a new event node c to $\hat{I}$ whose type is the same as c';
9 **for** *n' in c''s neighbor events and neighbor entities in S* **do**
10 **if** *n' has a matching node n in $\hat{I}$* **then**
11 Add a temporal or argument link between c and n in $\hat{I}$ according to the link between c' and n' in S;
12 **else**
13 **break**
14 **return** $\hat{I}$

3.5 Inference

The goal of inference is to complete an input instance event graph, i.e., predict all the missing nodes. This is similar to *set expansion* [10, 22, 33], which aims to expand a small set of seed elements into a complete set of elements that belong to the same semantic class. But note that there exists an intrinsic graph structure among elements in our problem. Therefore, we propose an inference algorithm specifically for schema-guided event graph completion.

The inference procedure is presented in Algorithm 1, which takes as input an incomplete instance graph I, the schema gaph S, and the trained model $f(e, I')$, then output the completed graph $\hat{I}$. The first step is to use the subgraph matching algorithm introduced in Section 3.1 to map I to a subgraph $I' \subseteq S$ (line 2), and the candidate event node set C is initialized as the set of nodes outside subgraph I' (line 3). Then we select a node from the candidate set C as the predicted missing event repeatedly (lines 4-13). At each iteration, we first identify the subset of the candidate set where nodes are neighbors of I' (i.e., $e \in C$ and $dist(e, I') = 1$), then select a node c' from this subset whose probability $f(c', I')$ is the highest, as the candidate event node at this iteration (line 5). The reason of restricting c' to be the neighbor of I' is that, c' can thus be linked directly to I' through temporal links, so that the instance graph after completion will still be connected. Based on the value of $f(c', I')$, we have the following two cases:

- If $f(c', I')$ is greater than a given threshold (line 6), such as 0.5, c' is taken as the predicted missing event at this iteration. We then update I' and C by adding c' to I' and removing c' from C, respectively (line 7), and update $\hat{I}$ by adding the missing event and its associated links to $\hat{I}$ (lines 8-11). Specifically, we add a new event node c to $\hat{I}$ whose type is the same as c' (line 8), then for each event and entity node $n' \in S$ that is connected to c', if n' has a matching node $n \in I$, we add a temporal link or argument link between

Dataset	Car-Bombings	IED-Bombings	Suicide-IED	Pandemic
# train/validation/test instance graphs	75 / 9 / 10	88 / 11 / 12	176 / 22 / 22	40 / 5 / 6
# train/validation/test samples	2,368 / 288 / 320	2,904 / 363 / 396	5,808 / 726 / 726	3,200 / 400 / 480
Corresponding schema name	Car-IED	General-IED		Disease-Outbreak
# event/entity nodes	32 / 134	33 / 140		102 / 17
# event-event/event-entity/entity-entity links	41 / 138 / 261	42 / 143 / 530		200 / 75 / 1

Table 1: Statistics of the four datasets and three event schemas.

c and n according to the link between c' and n' in S (lines 9-11). In other words, we "copy" the links associated to the predicted event from the schema to the instance graph.

- Otherwise (line 12), the scores of all nodes in the candidate set do not exceed the threshold (note that $f(c', I')$ is the largest among C). We can then terminate the inference procedure (line 13).

The inference loop is repeated until the candidate set C is empty or the probability $f(c', I')$ in the current loop is no larger than the threshold. Finally, $\hat{I}$ is returned as the completed event graph for I.

It is worth noting that missing events are predicted according to the ascending order of their distance to the subgraph I', which ensures that the original instance graph I is expanded over the schema in a natural, from-the-inside-out manner. In addition, note that the predicted missing event at one iteration is immediately added into I', which is then used to compute the probability f for the next iteration. Such a *bootstrapping* strategy allows the model to track the up-to-date instance graph and predict missing event nodes as many as possible.

4 EXPERIMENTS

In this section, we evaluate the proposed model, and present its performance on four datasets. The code and datasets are provided in the supplementary material.

4.1 Datasets

We conduct experiments on four event instance graph datasets: *Car-Bombings*, *IED-Bombings*, *Suicide-IED*, and *Pandemic*. The first three datasets are constructed by Li et al. [15], which consist of complex events related to IED bombing. The last dataset Pandemic is constructed by us. Specifically, we use RESIN [30], a cross-document information extraction and event tracking system, to process news articles mentioned in the references of Wikipedia articles related to pandemic, e.g., 2002-2004 SARS outbreak, then construct an instance event graph for each disease outbreak.

In addition, we use three complex event schemas for the four datasets. The first schema *Car-IED* describes the scenario of bombing caused by car IEDs that is detonated in an automobile or other vehicles. We use Car-IED schema for Car-Bombings dataset. The second schema *General-IED* describes the scenario of general IED bombing, which is used for IED-Bombings and Suicide-IED datasets. The last schema is *Disease-Outbreak*, which describes the spread of pandemics in a given population as well as the response of the authority, the public, etc. We use Disease-Outbreak schema for Pandemic dataset. The first two schemas are developed by Li et al. [15], while the last schema is manually curated by us. The statistics of the four datasets and the three schemas are presented in Table 1.

4.2 Baseline Methods

We compare our method with the following baseline methods:

- *AddAll*, which treats all events that exist in the schema but not exist in an instance graph as missing events. In other words, it treats all test samples as positive.
- *AddNeighbor*, which treats all events in the schema graph that are adjacent to the mapped subgraph of an instance graph as missing events. In other words, it treats a test sample (e, I') as positive if and only if e is the neighbor of at least one event in I'.
- *ID-MLP*, which concatenates the one-hot ID vector of e and the multi-hot ID vector of I' for the pair (e, I') as input, followed by an MLP to predict the probability that e is a missing event for I'.
- *Type-MLP*, which is similar to ID-MLP, but the input is the concatenated vector of the one-hot event type vector of e and the multi-hot event type vector of I' for the pair (e, I').
- *TransE* [1] is a classic knowledge graph completion method, which assumes that $\mathbf{h} + \mathbf{r} \approx \mathbf{t}$ for each triplet (h, r, t) in the knowledge graph, where h and t are the head and tail entities, respectively, and r is the relation.
- *RotatE* [23] is another state-of-the-art knowledge graph completion method similar to TransE, but it models entity and relation embeddings in the complex number space.

The implementation details of baselines are presented in Appendix A. In addition, we also conduct extensive ablation study and propose two reduced versions of our model, *SchemaEGC-Neighbor* and *SchemaEGC-Path*, which only use neighbor information and path information, respectively, to test the performance of the two components separately.

4.3 Experimental Setup

We evaluate our method on two tasks: *binary classification* and *graph completion*.

For the binary classification task, given an instance graph I in the test set, we first map I to the corresponding schema graph S and get the mapped subgraph $I' \subseteq S$. Then for each event $e \in I'$, we treat $(e, I' \backslash e)$ as a positive test sample, and for each event $e \in S \backslash I'$, we treat (e, I') as a negative test sample. We use *Accuracy* and *AUC* as the evaluation metrics.

For the graph completion task, given an instance graph I in the test set, we first randomly hide 10% of event nodes from I and treat the masked nodes as the ground-truth set, then run Algorithm 1 (the threshold in line 6 is set to 0.5) to complete the graph. Specifically, we predict the set of missing events using the remaining 90% of

Dataset	Car-Bombings		IED-Bombings		Suicide-IED		Pandemic	
Metrics	Accuracy	AUC	Accuracy	AUC	Accuracy	AUC	Accuracy	AUC
AddAll	0.553	0.500	0.353	0.500	0.404	0.500	0.190	0.500
AddNeighbor	0.594	0.569	0.586	0.641	0.609	0.645	0.648	0.568
ID-MLP	0.784 ± 0.032	0.889 ± 0.018	0.721 ± 0.008	0.857 ± 0.005	0.792 ± 0.007	0.883 ± 0.006	0.912 ± 0.006	0.963 ± 0.006
Type-MLP	0.802 ± 0.012	0.898 ± 0.008	0.724 ± 0.021	0.858 ± 0.009	0.799 ± 0.007	0.890 ± 0.004	0.914 ± 0.007	0.964 ± 0.004
TransE	0.795 ± 0.020	0.890 ± 0.019	0.733 ± 0.038	0.845 ± 0.031	0.784 ± 0.014	0.876 ± 0.009	0.905 ± 0.012	0.952 ± 0.010
RotatE	0.743 ± 0.040	0.860 ± 0.045	0.708 ± 0.026	0.865 ± 0.006	0.734 ± 0.013	0.851 ± 0.011	0.882 ± 0.015	0.940 ± 0.011
SchemaEGC	0.828 ± 0.017	**0.923** ± 0.003	**0.815** ± 0.012	**0.889** ± 0.002	**0.824** ± 0.003	**0.900** ± 0.005	**0.928** ± 0.005	**0.976** ± 0.001
SchemaEGC-Neighbor	**0.836** ± 0.014	0.921 ± 0.004	0.800 ± 0.021	0.888 ± 0.006	0.818 ± 0.006	0.898 ± 0.003	0.910 ± 0.003	0.956 ± 0.005
SchemaEGC-Path	0.809 ± 0.008	0.893 ± 0.007	0.795 ± 0.009	0.861 ± 0.004	0.818 ± 0.011	0.883 ± 0.008	0.919 ± 0.006	0.966 ± 0.002

Table 2: Mean and standard deviation of all methods on binary classification task. The best results are highlighted in bold.

Dataset	Car-Bombings		IED-Bombings		Suicide-IED		Pandemic	
Metrics	Jaccard Index	F1	Jaccard Index	F1	Jaccard Index	F1	Jaccard Index	F1
AddAll	0.173	0.278	0.083	0.150	0.099	0.173	0.032	0.061
AddNeighbor	0.165 ± 0.009	0.255 ± 0.019	0.109 ± 0.010	0.187 ± 0.016	0.124 ± 0.006	0.210 ± 0.011	0.047 ± 0.011	0.087 ± 0.020
ID-MLP	0.332 ± 0.019	0.466 ± 0.022	0.178 ± 0.033	0.282 ± 0.036	0.282 ± 0.024	0.391 ± 0.028	0.292 ± 0.050	0.380 ± 0.050
Type-MLP	0.387 ± 0.018	0.522 ± 0.022	0.187 ± 0.060	0.289 ± 0.075	0.336 ± 0.059	0.428 ± 0.053	0.303 ± 0.070	0.412 ± 0.064
TransE	0.349 ± 0.022	0.467 ± 0.029	0.196 ± 0.056	0.286 ± 0.063	0.302 ± 0.068	0.401 ± 0.064	0.273 ± 0.066	0.337 ± 0.082
RotatE	0.291 ± 0.065	0.418 ± 0.066	0.146 ± 0.037	0.235 ± 0.033	0.208 ± 0.024	0.301 ± 0.031	0.266 ± 0.058	0.319 ± 0.062
SchemaEGC	**0.458** ± 0.050	**0.592** ± 0.054	**0.348** ± 0.031	**0.483** ± 0.032	**0.449** ± 0.038	0.543 ± 0.050	**0.331** ± 0.043	**0.455** ± 0.073
SchemaEGC-Neighbor	0.420 ± 0.052	0.564 ± 0.054	0.328 ± 0.091	0.434 ± 0.103	0.444 ± 0.074	**0.555** ± 0.061	0.256 ± 0.009	0.389 ± 0.007
SchemaEGC-Path	0.416 ± 0.072	0.538 ± 0.067	0.288 ± 0.055	0.424 ± 0.071	0.378 ± 0.056	0.498 ± 0.055	0.311 ± 0.075	0.430 ± 0.078

Table 3: Mean and standard deviation of all methods on graph completion task. The best results are highlighted in bold.

event nodes in the graph, then compare the predicted set and the ground-truth set by calculating their *Jaccard Index* and $F1$ score.

We report the performance of our model on the test set when AUC on the validation set is maximized. Each experiment is repeated five times, and we report the mean and standard deviation of the results. The implementation details and hyperparameter settings are presented in Appendix A, and the sensitivity of our model to hyperparameter settings is presented in Appendix B.

4.4 Results

Comparison with Baselines. The results of binary classification task and graph completion task are presented in Table 2 and Table 3, respectively. Our method achieves the best performance on all datasets. Specifically, the AUC of our method SchemaEGC in binary classification task surpasses the best baseline by 2.5%, 2.4%, 1.0%, and 1.2%, respectively, on the four datasets, and the F1 of SchemaEGC in graph completion task surpasses the best baseline by 7.0%, 19.4%, 11.5%, and 4.3%, respectively, on the four datasets (all the numbers are absolute gains). We notice that the advantages of our method are much more significant in graph completion task, which demonstrate the superiority of our method in the real graph completion scenario.

We also observe that, in most cases, the two reduced versions of method SchemaEGC-Neighbor and SchemaEGC-Path already perform quite well and beat all the baseline methods. Combining them together usually leads to even better performance.

Impact of Schema Quality. Since our proposed method is based on event schemas, we study how the quality of schemas will impact the model performance. We randomly perturb the original Car-IED schema for Car-Bombings dataset, and present the model performance in Figure 3. The performance of the path module (SchemaEGC-path) is basically unchanged when the percentage of changed edges is small ($\leq 20\%$), but drops significantly when more edges in the schema are changed ($\geq 30\%$). In contrast, the neighbor module (SchemaEGC-neighbor) is more resistant to the noise in schema graph. As a result, the whole SchemaEGC model is able to basically maintain its performance even when half of the edges in the schema graph are changed.

4.5 Case Study

We conduct a case study on the predicted result of a test instance graph in Pandemic dataset, which describes the disease outbreak in a Chipotle restaurant. Figure 4 demonstrates the key part of the Disease-Outbreak schema, where dark red nodes and dark blue nodes denote instantiated events and entities (i.e., the mapped subgraph) according to the given instance graph, and striped nodes are missing events predicted by our model. The detailed description of the instantiated events and the predicted missing events is presented in Table 4. Our model can not only predict missing events but also provide the evidence for the prediction by analyzing the attention scores in the neighbor module and the path weights in the path module. For example, our model predicts that the No. 3 Eat event is missing from the instance graph, since it is close to two instantiated events: Exchange Goods and Illness (evidence of neighbors), and it can be linked to the mapped subgraph through a couple of high-weight paths such as (TEMP_REV), (meal, sold item), and (TEMP, TEMP) (evidence of paths). Both the evidence of neighbors and the evidence of paths can be used to provide explainability for the predicted results.

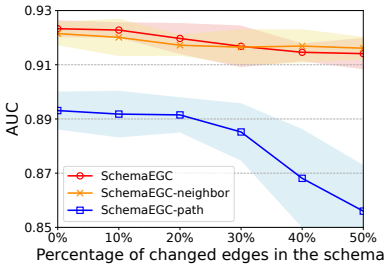


Figure 3: Impact of schema quality on Car-Bombings dataset.

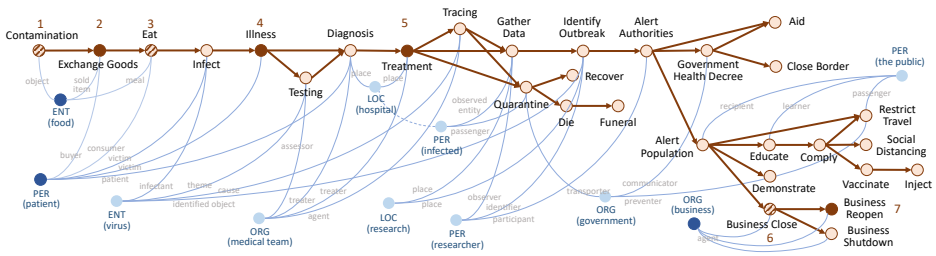


Figure 4: The Disease-Outbreak schema. Dark red nodes and dark blue nodes denote instantiated events and entities, respectively, and striped nodes are missing events predicted by our model. See Table 4 for more detailed description.

ID	Event type	Description	Event arguments	Evidence of neighbors	Evidence of paths
1	*Contamination*	The three chicken tacos are contaminated in the restaurant	Thing: Three chicken tacos	2	(TEMP) (object, sold item) (object, meal, TEMP_REV)
2	**Exchange Goods**	Filip Syzller purchased three chicken tacos at the Powell Chipotle restaurant	Buyer: Filip Syzller Sold Item: Three chicken tacos Location: Powell Chipotle	-	-
3	*Eat*	Filip Syzller ate the chicken tacos that are contaminated	Consumer: Filip Syzller Meal: Three chicken tacos	2, 4	(TEMP_REV) (meal, sold item) (TEMP, TEMP) (consumer, victim) (consumer, patient, TEMP)
4	**Illness**	Filip Syzller became ill with nausea, headaches, and severe abdominal pain in Ohio	Patient: Filip Syzller Disease: Nausea/Headaches Location: Ohio	-	-
5	**Treatment**	Filip Syzller was cured via medical treatment	Patient: Filip Syzller	-	-
6	*Business Close*	Powell Chiptole was closed due to the disease outbreak	Agent: Powell Chipotle	7	(TEMP_REV) (agent, agent)
7	**Business Reopen**	Powell Chipotle reopened its restaurant	Agent: Powell Chipotle	-	-
8	Investigate	Chipotle Mexican Grill, Inc. investigated Powell Chipotle	Location: Powell Chipotle Examiner: Chipotle Mexican Grill	-	-
9	Announcement	The Delaware General Health Department cited Powell Chipotle for not maintaining the proper temperature for some food items	Announcer: Health Department Hearer: Powell Chipotle Location: Powell	-	-

Table 4: Case study on the predicted result for a complex event regarding the disease outbreak in a Chipotle restaurant. The bold events means that they are instantiated in the schema graph, the italic events means that they are the missing events predicted by our model, while the last two events cannot be matched to the schema. Evidence of neighbors and evidence of paths denote the neighbors and paths that are important for our model to make the prediction, respectively.

5 RELATED WORK

In this section, we discuss three lines of related work: event schema induction, knowledge graph completion, and graph matching.

Event Schema Induction. Event schema induction aims to automatically learn and induce event schemas from event instances. The first class of event schema induction methods are sequence-based, which takes event-event relations into account, and orders event structures into sequences [8, 28, 29]. Since they fail to capture the multi-dimensional evolution of real-world complex events, i.e., an event can be preceded or followed by multiple events, researchers propose graph-based schema induction methods, which use graphs to formulate event schemas [15, 16, 27]. For example, Li et al. [15] train an auto-regressive graph generation model on instance event graphs and then generate the event schema event by event. Our work differs from these methods in that their goal is to learn event schemas, while we focus on the downstream task of event schema, i.e., event graph completion. Nonetheless, these methods can be used to output event schemas for our schema-guided model.

Knowledge Graph Completion. Event graphs are a type of heterogeneous graphs, which is conceptually related to traditional entity-centric knowledge graphs. Many knowledge graph completion methods are embedding-based [1, 11, 23, 31, 35], which learn an embedding vector for each entity and relation by minimizing a predefined loss function on all triplets. Such methods have the advantage that they consider the structural context of a given entity in the KG but they fail to capture the multiple relationships (paths)

between the head and the tail entity. In contrast, the second class of methods is rule-based [5, 21, 32], which aims to learn general logical rules from knowledge graphs by modeling paths between the head and the tail entities. However, a significant drawback of these methods is that meaningful rules are usually very sparse, which limits their capability of predicting missing relations that are not covered by known rules. Similar to Wang et al. [25], our method can be seen as combining the methodology of the two classes of methods, but our method is specifically designed for event graphs.

Graph Matching. Due to the NP-hardness of exact graph matching algorithms [3], neural graph matching is proposed to solve the problem by machine learning and deep neural networks [2, 17, 18, 20]. For example, Graph Matching Networks [17] learns the node matching relations between two graphs by an attention-based graph neural networks. However, the graphs in their study and ours are quite different, because the nodes in event graphs have type information and they should be strictly matched, which will greatly reduce the matching space. Therefore, our proposed subgraph matching algorithm fully utilizes the event type information and runs much faster than traditional or neural network based methods.

6 CONCLUSION AND FUTURE WORK

We propose a schema-guided method for event graph completion, which overcomes the drawbacks of existing graph completion methods and enables the model to predict new events that are missing in the instance event graphs. We consider two factors when modeling the event schema graph, i.e. neighbors and paths, to fully capture its high-order topological and semantic information. Experimental results on four datasets and three schemas demonstrate that our method achieves state-of-the-art performance on event graph completion task. Moreover, it is resistant to noise in the schema and exhibits high explainability for the prediction results.

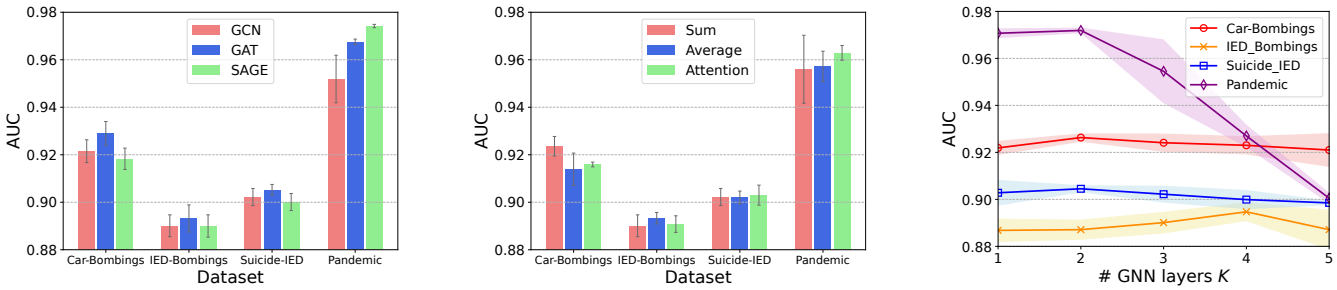
Note that in this work, we focus on predicting missing nodes for event graphs. Since the links in event graphs could also be noisy, a future direction is therefore to refine links in event graphs based on event schemas. In addition, in some event schemas, events are organized in a hierarchical manner (e.g., a pandemic event may include multiple episodes such as Authority Response and Society Response, and each episode may include multiple atomic events, e.g., Authority Response includes Restrict Travel and Close Border). We plan to utilize such hierarchy to improve the model performance.

7 ACKNOWLEDGEMENT

This research is based upon work supported in part by U.S. DARPA KAIROS Program No. FA8750-19-2-1004. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of DARPA, or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for governmental purposes notwithstanding any copyright annotation therein.

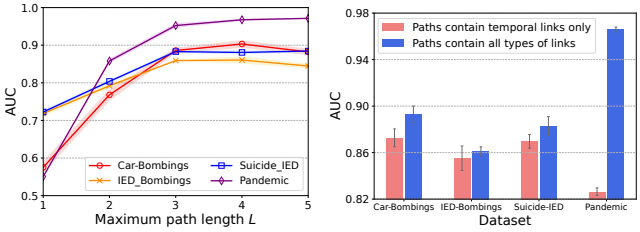
REFERENCES

[1] Antoine Bordes, Nicolas Usunier, Alberto Garcia-Duran, Jason Weston, and Oksana Yakhnenko. 2013. Translating embeddings for modeling multi-relational data. *NeurIPS* 26 (2013).
[2] Minsu Cho, Jian Sun, Olivier Duchenne, and Jean Ponce. 2014. Finding matches in a haystack: A max-pooling strategy for graph matching in the presence of outliers. In *CVPR*. 2083–2090.
[3] Donatello Conte, Pasquale Foggia, Carlo Sansone, and Mario Vento. 2004. Thirty years of graph matching in pattern recognition. *International journal of pattern recognition and artificial intelligence* 18, 03 (2004), 265–298.
[4] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. Bert: Pre-training of deep bidirectional transformers for language understanding. In *NAACL-HLT*.
[5] Luis Galárraga, Christina Teflioudi, Katja Hose, and Fabian M Suchanek. 2015. Fast rule mining in ontological knowledge bases with AMIE++. *The VLDB Journal* 24, 6 (2015), 707–730.
[6] Goran Glavaš and Jan Šnajder. 2015. Construction and evaluation of event graphs. *Natural Language Engineering* 21, 4 (2015), 607–652.
[7] Rishab Goel, Seyed Mehran Kazemi, Marcus Brubaker, and Pascal Poupart. 2020. Diachronic embedding for temporal knowledge graph completion. In *AAAI*, Vol. 34. 3988–3995.
[8] Mark Granroth-Wilding and Stephen Clark. 2016. What happens next? event prediction using a compositional neural network model. In *AAAI*, Vol. 30.
[9] William L Hamilton, Rex Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. In *NeurIPSS*. 1025–1035.
[10] Jiaxin Huang, Yiqing Xie, Yu Meng, Jiaming Shen, Yunyi Zhang, and Jiawei Han. 2020. Guiding corpus-based set expansion by auxiliary sets generation and co-expansion. In *WWW*. 2188–2198.
[11] Seyed Mehran Kazemi and David Poole. 2018. Simple embedding for link prediction in knowledge graphs. *NeurIPS* 31 (2018).
[12] Diederik P Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. In *ICLR*.
[13] Thomas N Kipf and Max Welling. 2017. Semi-supervised classification with graph convolutional networks. In *ICLR*.
[14] Kai Lei, Meng Qin, Bo Bai, Gong Zhang, and Min Yang. 2019. Gcn-gan: A non-linear temporal link prediction model for weighted dynamic networks. In *INFOCOM*. IEEE, 388–396.
[15] Manling Li, Sha Li, Zhenhailong Wang, Lifu Huang, Kyunghyun Cho, Heng Ji, Jiawei Han, and Clare Voss. 2021. The Future is not One-dimensional: Complex Event Schema Induction by Graph Modeling for Event Prediction. In *EMNLP*. 5203–5215.
[16] Manling Li, Qi Zeng, Ying Lin, Kyunghyun Cho, Heng Ji, Jonathan May, Nathanael Chambers, and Clare Voss. 2020. Connecting the dots: Event graph schema induction with path language modeling. In *EMNLP*. 684–695.
[17] Yujia Li, Chenjie Gu, Thomas Dullien, Oriol Vinyals, and Pushmeet Kohli. 2019. Graph matching networks for learning the similarity of graph structured objects. In *ICML*. PMLR, 3835–3845.
[18] Zhaoyu Lou, Jiaxuan You, Chengtao Wen, Arquimedes Canedo, Jure Leskovec, et al. 2020. Neural subgraph matching. *arXiv preprint arXiv:2007.03092* (2020).
[19] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. 2019. Pytorch: An imperative style, high-performance deep learning library. *NeurIPS* 32 (2019).
[20] Hermina Petric Maretic, Mireille El Gheche, Giovanni Chierchia, and Pascal Frossard. 2019. GOT: an optimal transport framework for graph comparison. *NeurIPS* 32 (2019).
[21] Ali Sadeghian, Mohammadreza Armandpour, Patrick Ding, and Daisy Zhe Wang. 2019. Drum: End-to-end differentiable rule mining on knowledge graphs. In *NeurIPS*, Vol. 32.
[22] Jiaming Shen, Zeqiu Wu, Dongming Lei, Jingbo Shang, Xiang Ren, and Jiawei Han. 2017. Setexpan: Corpus-based set expansion via context feature selection and rank ensemble. In *ECML-PKDD*. Springer, 288–304.
[23] Zhiqing Sun, Zhi-Hong Deng, Jian-Yun Nie, and Jian Tang. 2018. RotatE: Knowledge Graph Embedding by Relational Rotation in Complex Space. In *ICLR*.
[24] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. 2018. Graph Attention Networks. In *ICLR*.
[25] Hongwei Wang, Hongyu Ren, and Jure Leskovec. 2021. Relational message passing for knowledge graph completion. In *KDD*. 1697–1707.
[26] Hongwei Wang, Fuzheng Zhang, Min Hou, Xing Xie, Minyi Guo, and Qi Liu. 2018. Shine: Signed heterogeneous information network embedding for sentiment link prediction. In *WSDM*. 592–600.
[27] Lilian DA Wanzare, Alessandra Zarcone, Stefan Thater, and Manfred Pinkal. 2016. A crowdsourced database of event sequence descriptions for the acquisition of high-quality script knowledge. (2016).
[28] Noah Weber, Niranjan Balasubramanian, and Nathanael Chambers. 2018. Event representations with tensor-based compositions. In *AAAI*, Vol. 32.
[29] Noah Weber, Rachel Rudinger, and Benjamin Van Durme. 2020. Causal Inference of Script Knowledge. In *EMNLP*. 7583–7596.



(a) GNN type **(b) READOUT function** **(c) # GNN layers K**

Figure 5: Hyperparameter sensitivity of SchemaEGC-Neighbor w.r.t. GNN type, READOUT function, and the number of GNN layers K.



(a) Maximum path length L **(b) Path type**

Figure 6: Hyperparameter sensitivity of SchemaEGC-Path w.r.t. the maximum path length L and path type.

[30] Haoyang Wen, Ying Lin, Tuan Lai, Xiaoman Pan, Sha Li, Xudong Lin, Ben Zhou, Manling Li, Haoyu Wang, Hongming Zhang, et al. 2021. RESIN: A Dockerized Schema-Guided Cross-document Cross-lingual Cross-media Information Extraction and Event Tracking System. In *NAACL-HLT*. 133–143.

[31] Bishan Yang, Scott Wen-tau Yih, Xiaodong He, Jianfeng Gao, and Li Deng. 2015. Embedding Entities and Relations for Learning and Inference in Knowledge Bases. In *ICLR*.

[32] Fan Yang, Zhilin Yang, and William W Cohen. 2017. Differentiable learning of logical rules for knowledge base reasoning. In *NeurIPS*, Vol. 30.

[33] Manzil Zaheer, Satwik Kottur, Siamak Ravanbhakhsh, Barnabás Póczos, Ruslan Salakhutdinov, and Alexander J Smola. 2017. Deep Sets. In *NeurIPS*. 3394–3404.

[34] Muhan Zhang and Yixin Chen. 2018. Link prediction based on graph neural networks. *NeurIPS* 31 (2018), 5165–5175.

[35] Shuai Zhang, Yi Tay, Lina Yao, and Qi Liu. 2019. Quaternion Knowledge Graph Embeddings. In *NeurIPS*. 2735–2745.

APPENDIX

A Implementation Details

Baseline Methods. For ID-MLP and Type-MLP, we use an MLP with one hidden layer of 100 units as the prediction model. For TransE and RotatE, we first use them to learn the embedding of each node in the schema graph, then average the embeddings of nodes in a subgraph as the subgraph embedding. For an input pair (e, I'), their embeddings are concatenated, followed by an MLP with one hidden layer of 100 units to predict the missing probability. The code of TransE and RotatE is from https://github.com/DeepGraphLearning/KnowledgeGraphEmbedding. We set the dimension of entity embedding to 256 and keep other hyperparameters as default.

Our Method. Our method is implemented in PyTorch [19]. We use GCN [13] as the implementation of GNN in the neighbor module. The number of GNN layers K is 3, and the dimension of hidden layers is 256. We use sum as the READOUT function. The maximum path length L is 4 in the path module. $\text{MLP}_{neighbor}$ and MLP_{path} are both MLPs with one hidden layer of dimension 256. We train the model for 20 epochs with a batch size of 128, using Adam [12] optimizer with a learning rate of 0.005. The above hyperparameters are determined by maximizing the AUC on the validation set of Car-Bombings, and kept unchanged for all dataset. The result of hyperparameter sensitivity is provided in Appendix B. The search space of hyperparameters are as follows:

- GNN type: {GCN, GAT, SAGE};
- The number of GNN layers: $\{1, 2, 3, 4, 5\}$;
- The dimension of GNN hidden layers: $\{32, 64, 128, 256, 512\}$;
- READOUT function: {sum, average, attention};
- The maximum path length: $\{1, 2, 3, 4, 5\}$;
- The dimension of hidden layer in $\text{MLP}_{neighbor}$ and MLP_{path}: $\{32, 64, 128, 256, 512\}$;
- Batch size: $\{32, 64, 128, 256, 512\}$;
- Learning rate: $\{0.0005, 0.001, 0.005, 0.01, 0.05, 0.1\}$.

B Hyperparameter Sensitivity

We study the sensitivity of our model to several key hyperparameters to provide better understanding on the proposed model and investigate its robustness.

For the neighbor module of SchemaEGC, we study the impact of GNN type, READOUT function, and the number of GNN layers K on the model performance. As shown in Figure 5a, we use three GNNs in the neighbor module: GCN [13], GAT [24], and SAGE [9]. The number of GNN layers is 3, and the dimension of hidden layers is 256 (for GAT, the number of attention heads is 16 and the dimension of each attention head is 16). The result shows that GAT achieves the best performance in the three IED-related scenarios while SAGE performs the best in the pandemic scenario. Figure 5b demonstrates that attention is a good choice for the READOUT function, but it does not perform universally the best on all datasets. Figure 5c shows that our model usually achieves the best performance when the number of GNN layers is $2 \sim 4$. Moreover, Pandemic dataset is much more sensitive to the number of GNN layers.

For the path module of SchemaEGC, we study the impact of the maximum path length and the path type. From Figure 6a it is clear that our model is extremely sensitive to the maximum path length L, since the AUC increases by $0.2 \sim 0.4$ as L goes from 1 to 5. But the marginal improvement is diminishing when $L \geq 3$ due to the issue of overfitting. Figure 6b demonstrates that considering all the three types of links for the path module works better than only considering event-event temporal links.