Schema-Guided Event Graph Completion

Hongwei Wang HONGWEIW@ILLINOIS.EDU
University of Illinois Urbana-Champaign

Zixuan Zhang ZIXUAN11@ILLINOIS.EDU
University of Illinois Urbana-Champaign

Sha Li SHAL2@ILLINOIS.EDU
University of Illinois Urbana-Champaign

Jiawei Han HANJ@ILLINOIS.EDU
University of Illinois Urbana-Champaign

Yizhou Sun YZSUN@CS.UCLA.EDU
University of California, Los Angeles

Hanghang Tong HTONG@ILLINOIS.EDU
University of Illinois Urbana-Champaign

Joseph P. Olive JOSEPH.OLIVE.CTR@DARPA.MIL
JP Olive S&T Management LLC

Heng Ji HENGJI@ILLINOIS.EDU
University of Illinois Urbana-Champaign

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 event graphs. Moreover, they can only predict missing edges rather than missing nodes. In this work, we utilize *event schemas*, a type of generalized representation that describes the stereotypical structure of event graphs, to address these issues. Our schema-guided event graph completion approach first maps an instance event graph to a schema subgraph. Then it predicts whether a candidate event node in the schema graph should be instantiated by characterizing two aspects of local topology: *neighbors* of both the candidate node and the schema subgraph, and *paths* that connect the candidate node and the schema subgraph. The neighbor module and the path module are later combined together for the final prediction. 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. The code and datasets are available at https://github.com/hwwang55/SchemaEGC.

1. Introduction

Event graphs [Glavaš and Šnajder, 2015] are structured representation of real-world complex events. 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. Event graphs can act as a useful tool to help readers quickly understand and untangle the progression of complex events. 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 events and entities, respectively.

A practical issue of automatically constructed event graphs is that they are often incomplete and noisy, due to reporting bias in the source documents, limited coverage of existing ontologies which information extraction models are trained on, or the imperfect performance of existing information extraction methods [Lin et al., 2020, Li et al., 2021b]. For example, in the instance graph in Figure 1, the INJURE:0 event probably has an argument PLACE linked to an 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 [Zhang and Chen, 2018, Wang et al., 2018, Lei et al., 2019] or graph completion [Zhang et al., 2019, Goel et al., 2020, Wang et al., 2021] 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 or even millions of nodes), but an event graph dataset usually 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, existing link prediction or graph completion methods are incapable of predicting the next-step event after IDENTIFY:0.

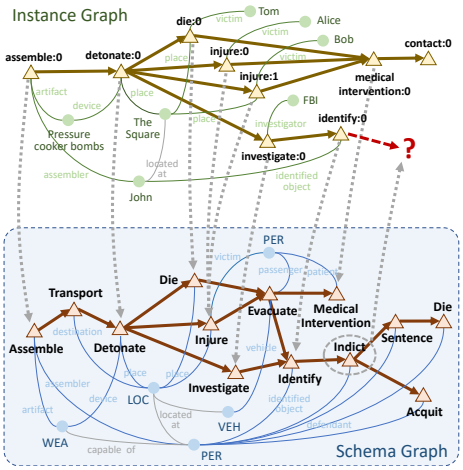


Figure 1: An example of schema-guided event graph completion. An (incomplete) event instance graph is shown on the top half and the corresponding schema is shown at the bottom. By matching the instance graph against the schema (dotted lines), we can predict that there should be an INDICT event following the IDENTIFY:0 event.

To address the limitations of existing methods, a promising solution is to use 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 [Granroth-Wilding and Clark, 2016, Weber et al., 2018, 2020, Li et al., 2020, 2021a] from a large collection of historical event graphs, or manually curated by human[1]. 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

1. Schemas can be curated by human efficiently using curation tools [Mishra et al., 2021]. The high-quality human curated schemas are available for a wide variety of newsworthy scenarios.

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 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.

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 alignment 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. After the 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: (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., 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 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. 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.

We conduct 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, our F1 score surpasses the best baseline method (MLP-based method, TransE, or RotatE) 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

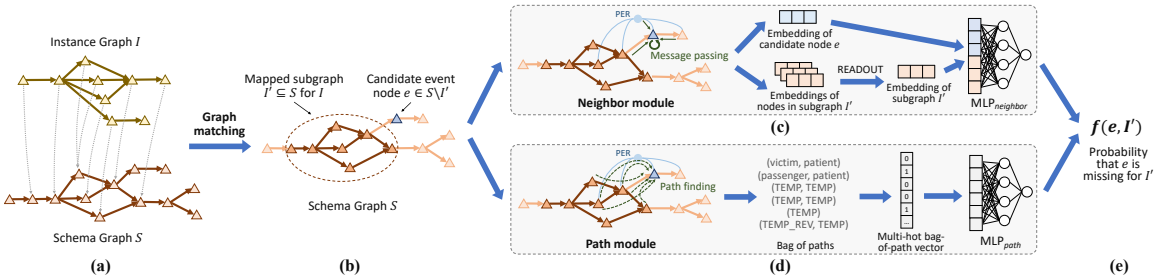


Figure 2: Illustration of the model architecture. (a) The input instance graph and schema graph; (b) Matching the instance graph to a subgraph of schema. (c) The neighbor module; (d) The path module; (e) Combining the two modules and output final prediction.

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, and we use "TEMP" to denote the type of temporal links; (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; (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. We also use function $\tau(\cdot)$ to denote the type of a link. Specifically, $\tau(\langle e_i, e_j \rangle) = \text{TEMP}$, $\tau(\langle e_i, a, v_j \rangle) = a$, 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.

3. Our Approach

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.

Subgraph matching is usually defined as a 0-1 integer programming problem [Cho et al., 2014], which is NP-complete (See Appendix A for details). To improve the time efficiency, 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:

$$E_i = \{e_j \in S \mid \tau(e_j) = \tau(e_i)\}, \tag{1}$$

If $|E_i| = 0$, e_i will not be matched to any event node in S; If $|E_i| = 1$, we end up with a unique node in S, which is taken as the matching node for e_i; Otherwise ($|E_i| > 1$), there are multiple 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. In this case, we design an additional stage for event matching, which is based on the similarity of neighboring 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:

$$\mathcal{P}_S(e_j)=\{\tau(e_k) \mid \langle e_k, e_j\rangle \in S\},\ \mathcal{F}_S(e_j)=\{\tau(e_k) \mid \langle e_j, e_k\rangle \in S\},\ \mathcal{A}_S(e_j)=\{a \mid \langle e_j, a, v_k\rangle \in S\}.$$

We can also identify the above three sets for e_i in instance graph I, i.e., $\mathcal{P}_I(e_i)$, $\mathcal{F}_I(e_i)$, and $\mathcal{A}_I(e_i)$. Then we calculate the Jaccard index[2] J of the three corresponding set pairs, and select the node with the highest total Jaccard index as the matching result:

$$e^* = \arg\max_{e_j \in E_i}\ J\big(\mathcal{P}_S(e_j), \mathcal{P}_I(e_i)\big) + J\big(\mathcal{F}_S(e_j), \mathcal{F}_I(e_i)\big) + J\big(\mathcal{A}_S(e_j), \mathcal{A}_I(e_i)\big). \quad (2)$$

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. The complexity analysis on the two-step matching scheme is presented in Appendix B.

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 a node by aggregating representations of its neighbors. For example, in Graph Convolutional Networks (GCN) [Kipf and Welling, 2017], the k-th layer is

$$\mathbf{h}_i^k = \sigma\Big(\mathbf{W}^k \sum\nolimits_{j \in \mathcal{N}(i)\cup\{i\}} \frac{1}{\sqrt{|\mathcal{N}(i)| \cdot |\mathcal{N}(j)|}} \mathbf{h}_j^{k-1} + \mathbf{b}^k\Big), \quad (3)$$

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), $\mathcal{N}(i)$ is the set of nodes directly connected to i, $\mathbf{W}^k$ 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 E.

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 $\{\mathbf{h}_{e_i}^K\}_{e_i \in I'}$, respectively. Then, a readout function

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

is used to aggregate event embeddings in I' and output the embedding of I':

$$\mathbf{h}_{I'}^{K} = \text{READOUT}\left(\left\{\mathbf{h}_{e_i}^{K}\right\}_{e_i \in I'}\right). \tag{4}$$

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 w.r.t. the given event e: $\mathbf{h}_{I'}^{K} = \sum_{e_i \in I'} \beta_i \mathbf{h}_{e_i}^{K}$, where $\beta_i = \mathbf{h}_{e_i}^{K\top} \mathbf{h}_{e}^{K} / \sum_{e_i \in I'} \mathbf{h}_{e_i}^{K\top} \mathbf{h}_{e}^{K}$ is the attention weight. We report the performance of different READOUT functions in Appendix E.

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':

$$p_{neighbor}(e, I') = \text{MLP}_{neighbor}\left(\left[\mathbf{h}_{e}^{K}, \mathbf{h}_{I'}^{K}\right]\right). \tag{5}$$

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. 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. Specifically, a path connecting two nodes s and t is a sequence of nodes and edges: $s \xrightarrow{\langle s,i \rangle} i \xrightarrow{\langle i,j \rangle} j \cdots k \xrightarrow{\langle k,t \rangle} t$, where $\langle i, j \rangle$ is the edge connecting node i and j, and each node in the path is unique[3]. In this work, we use the types of edges in a path to represent the path, i.e., $\big(\tau(\langle s,i \rangle), \tau(\langle i,j \rangle), \cdots, \tau(\langle k,t \rangle)\big)$. We use $\mathcal{P}_{s \to t}^{\leq L}$ 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., $\mathcal{P}_{\text{Transport} \to \text{Detonate}}^{\leq 2} = \{(\text{TEMP}), (\text{destination}, \text{place})\}$.

For a 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'):

$$\mathcal{P}_{e \to I'}^{\leq L} = \bigcup\nolimits_{e_i \in I'} \mathcal{P}_{e \to e_i}^{\leq L}, \tag{6}$$

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

$$p_{path}(e, I') = \text{MLP}_{path}\left(\mathbf{p}_{e \to I'}^{\leq L}\right). \tag{7}$$

Finally, our predicting function $f(e, I')$ is implemented by combining the output of the neighbor module in Eq. (5) and the path module in Eq. (7):

$$f(e, I') = \big(p_{neighbor}(e, I') + p_{path}(e, I')\big) \,/\, 2. \tag{8}$$

3. 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.

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:

$$L = \frac{1}{|\mathcal{I}|} \sum\nolimits_{I \in \mathcal{I}} \frac{1}{|S|} \Big(\sum\nolimits_{e \in I'} \mathcal{C}\Big(f(e, I' \backslash e), 1\Big) + \sum\nolimits_{e \in S \backslash I'} \mathcal{C}\Big(f(e, I'), 0\Big) \Big), \tag{9}$$

where $\mathcal{I}$ is the set of training instance graphs, I' is a subgraph of S that I is mapped to, and $\mathcal{C}$ is cross-entropy loss. In Eq. (9), the first and the second term are the loss functions for positive and negative samples, respectively. Note that the training data constructed by Eq. (9) may be unbalanced. In this case, we can use downsampling to re-balance the dataset.

During the inference stage, the goal is to complete an input instance event graph, i.e., predict all its missing nodes. We discuss the details of the inference algorithm in Appendix C.

4. Experiments

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. [2021a], which consist of complex events related to IED bombing. The last dataset Pandemic is constructed by us. Specifically, we use RESIN [Wen et al., 2021, Du et al., 2022], 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 and COVID-19, then construct an instance event graph for each disease outbreak.

In addition, we use three complex event schemas for these four datasets. The first schema *Car-IED* describes the scenario of bombings caused by car IEDs that are detonated in an automobile or other vehicles. The second schema *General-IED* describes the scenario of general IED bombing. The last schema is *Disease-Outbreak*, which describes the spread of diseases in a given population as well as the response of the authority, the public, etc. The first two schemas are developed by Li et al. [2021a], while the last schema is manually curated by us. The statistics of datasets and schemas is presented in Table 1.

4.2 Baseline Methods

We compare our method with the following baseline methods: (1) *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. (2) *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

Dataset	Car-Bombings	IED-Bombings	Suicide-IED	Pandemic
# train/val/test instance graphs	75 / 9 / 10	88 / 11 / 12	176 / 22 / 22	40 / 5 / 6
# train/val/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
# ev-ev/ev-en/en-en links	41 / 138 / 261	42 / 143 / 530		200 / 75 / 1

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

is the neighbor of at least one event in I'. (3) *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'. (4) *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'). (5) *TransE* [Bordes et al., 2013] 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. (6) *RotatE* [Sun et al., 2018] 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 D. In addition, we also conduct extensive ablation study and propose two reduced versions of our model, *SEGC-neighbor* and *SEGC-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 binary classification task, given an instance graph I in the test set, we first map I to 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* (Area Under the Curve) as the evaluation metrics. For graph completion task, given an instance graph I in the test set, we first randomly hide 10% 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 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 D, and the sensitivity of our model to hyperparameters is presented in Appendix E.

4.4 Results

The result of binary classification task and graph completion task are presented in Table 2 and 3, respectively. Our method achieves the best performance on all datasets. Specifically,

Dataset	Car-Bombings		IED-Bombings		Suicide-IED		Pandemic	
Metrics	Accuracy	AUC	Accuracy	AUC	Accuracy	AUC	Accuracy	AUC
AddAll	55.3	50.0	35.3	50.0	40.4	50.0	19.0	50.0
AddNeighbor	59.4	56.9	58.6	64.1	60.9	64.5	64.8	56.8
ID-MLP	78.4 ± 3.2	88.9 ± 1.8	72.1 ± 0.8	85.7 ± 0.5	79.2 ± 0.7	88.3 ± 0.6	91.2 ± 0.6	96.3 ± 0.6
Type-MLP	80.2 ± 1.2	89.8 ± 0.8	72.4 ± 2.1	85.8 ± 0.9	79.9 ± 0.7	89.0 ± 0.4	91.4 ± 0.7	96.4 ± 0.4
TransE	79.5 ± 2.0	89.0 ± 1.9	73.3 ± 3.8	84.5 ± 3.1	78.4 ± 1.4	87.6 ± 0.9	90.5 ± 1.2	95.2 ± 1.0
RotatE	74.3 ± 4.0	86.0 ± 4.5	70.8 ± 2.6	86.5 ± 0.6	73.4 ± 1.3	85.1 ± 1.1	88.2 ± 1.5	94.0 ± 1.1
SEGC	82.8 ± 1.7	**92.3** ± 0.3	**81.5** ± 1.2	**88.9** ± 0.2	**82.4** ± 0.3	**90.0** ± 0.5	**92.8** ± 0.5	**97.6** ± 0.1
SEGC-neighbor	**83.6** ± 1.4	92.1 ± 0.4	80.0 ± 2.1	88.8 ± 0.6	81.8 ± 0.6	89.8 ± 0.3	91.0 ± 0.3	95.6 ± 0.5
SEGC-path	80.9 ± 0.8	89.3 ± 0.7	79.5 ± 0.9	86.1 ± 0.4	81.8 ± 1.1	88.3 ± 0.8	91.9 ± 0.6	96.6 ± 0.2

Table 2: Mean and standard deviation (in %) of all methods on binary classification task.

Dataset	Car-Bombings		IED-Bombings		Suicide-IED		Pandemic	
Metrics	Jaccard	F1	Jaccard	F1	Jaccard	F1	Jaccard	F1
AddAll	17.3	27.8	8.3	15.0	9.9	17.3	3.2	6.1
AddNeighbor	16.5 ± 0.9	25.5 ± 1.9	10.9 ± 1.0	18.7 ± 1.6	12.4 ± 0.6	21.0 ± 1.1	4.7 ± 1.1	8.7 ± 2.0
ID-MLP	33.2 ± 1.9	46.6 ± 2.2	17.8 ± 3.3	28.2 ± 3.6	28.2 ± 2.4	39.1 ± 2.8	29.2 ± 5.0	38.0 ± 5.0
Type-MLP	38.7 ± 1.8	52.2 ± 2.2	18.7 ± 6.0	28.9 ± 7.5	33.6 ± 5.9	42.8 ± 5.3	30.3 ± 7.0	41.2 ± 6.4
TransE	34.9 ± 2.2	46.7 ± 2.9	19.6 ± 5.6	28.6 ± 6.3	30.2 ± 6.8	40.1 ± 6.4	27.3 ± 6.6	33.7 ± 8.2
RotatE	29.1 ± 6.5	41.8 ± 6.6	14.6 ± 3.7	23.5 ± 3.3	20.8 ± 2.4	30.1 ± 3.1	26.6 ± 5.8	31.9 ± 6.2
SEGC	**45.8** ± 5.0	**59.2** ± 5.4	**34.8** ± 3.1	**48.3** ± 3.2	**44.9** ± 3.8	54.3 ± 5.0	**33.1** ± 4.3	**45.5** ± 7.3
SEGC-neighbor	42.0 ± 5.2	56.4 ± 5.4	32.8 ± 9.1	43.4 ± 10.3	44.4 ± 7.4	**55.5** ± 6.1	25.6 ± 0.9	38.9 ± 0.7
SEGC-path	41.6 ± 7.2	53.8 ± 6.7	28.8 ± 5.5	42.4 ± 7.1	37.8 ± 5.6	49.8 ± 5.5	31.1 ± 7.5	43.0 ± 7.8

Table 3: Mean and standard deviation (in %) of all methods on graph completion task.

the AUC of our method SEGC 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 SEGC 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 SEGC-neighbor and SEGC-path already perform quite well and beat all the baseline methods. Combining them together usually leads to even better performance.

We also examine the impact of schema quality on our model performance, which is discussed in Appendix F.

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 3 demonstrates the key part of the Disease-Outbreak schema. The detailed description of the instantiated events and the predicted missing events are shown 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 some high-weight paths such as (TEMP_REV) and (meal, sold item) (evidence of paths). Both the evidence of

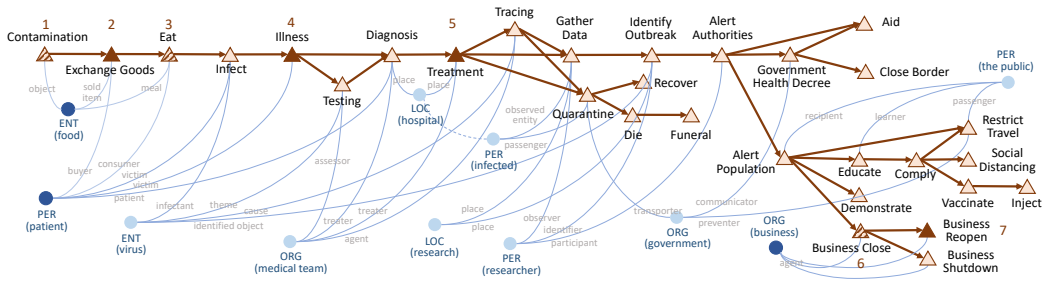


Figure 3: Case study on Disease-Outbreak schema. Dark red/blue nodes denote instantiated events/ entities, and striped nodes are missing events predicted by our model.

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 a disease outbreak event in a Chipotle restaurant. The **bold** events indicate that they are instantiated in the schema graph, the *italic* events indicate 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.

neighbors and the evidence of paths can be used to provide explainability for the predicted results.

5. Related Work

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

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 [Granroth-Wilding and Clark, 2016, Weber et al., 2018, 2020, Lyu et al., 2020, Yang et al., 2021, Zhang et al., 2021, Zhou et al., 2022]. 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 [Wanzare et al., 2016, Li et al., 2020, 2021a, Jin et al., 2022]. For example, Li et al. [2021a] train an auto-regressive graph generation model on instance event graphs and then generate the event schema by aggregating and generalizing all events. Our work differs from these methods in that their goal is to learn event schemas, while we focus on the downstream application of event schema, i.e., event graph completion. Nonetheless, these methods can be used to generate event schemas for our schema-guided prediction model.

Event graph is a type of heterogeneous graphs, which is conceptually related to traditional entity-centric knowledge graphs. Many knowledge graph completion methods are embedding-based [Bordes et al., 2013, Yang et al., 2015, Kazemi and Poole, 2018, Zhang et al., 2019], which learn an embedding vector for each entity and relation by minimizing a predefined loss function on all triplets. Such methods have the advantage by considering the structural context of a given entity in the KG, but they fail to capture multiple relations (paths) between entities. In contrast, the second class of methods is rule-based [Galárraga et al., 2015, Sadeghian et al., 2019, Yang et al., 2017], which aims to learn general logical rules from knowledge graphs by modeling paths between the head and 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. [2021], our method can be seen as combining the methodology of the two classes of methods, but our method is specifically designed for event graphs.

6. Conclusion and Future Work

We propose a new schema-guided method for event graph completion, which overcomes the drawbacks of existing graph completion methods and enables the model to predict missing events in instance event graphs. We consider neighbors and paths when modeling the event schema graph 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, and contain logical relations and goal predictions. We plan to utilize such rich knowledge to further improve the model performance.

Acknowledgement

We thank the anonymous reviewers helpful suggestions. This research is based upon work supported 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

Antoine Bordes, Nicolas Usunier, Alberto Garcia-Duran, Jason Weston, and Oksana Yakhnenko. Translating embeddings for modeling multi-relational data. *NeurIPS*, 26, 2013.

Minsu Cho, Jian Sun, Olivier Duchenne, and Jean Ponce. Finding matches in a haystack: A max-pooling strategy for graph matching in the presence of outliers. In *CVPR*, pages 2083–2090, 2014.

Xinya Du, Zixuan Zhang, Sha Li, Pengfei Yu, Hongwei Wang, Tuan Manh Lai, Xudong Lin, Ziqi Wang, Iris Liu, Ben Zhou, et al. Resin-11: Schema-guided event prediction for 11 newsworthy scenarios. In *Proc. 2022 Annual Conference of the North American Chapter of the Association for Computational Linguistics (NAACL2022) System Demonstration Track*, 2022.

Luis Galárraga, Christina Teflioudi, Katja Hose, and Fabian M Suchanek. Fast rule mining in ontological knowledge bases with amie++. *The VLDB Journal*, 24(6):707–730, 2015.

Goran Glavaš and Jan Šnajder. Construction and evaluation of event graphs. *Natural Language Engineering*, 21(4):607–652, 2015.

Rishab Goel, Seyed Mehran Kazemi, Marcus Brubaker, and Pascal Poupart. Diachronic embedding for temporal knowledge graph completion. In *AAAI*, volume 34, pages 3988–3995, 2020.

Mark Granroth-Wilding and Stephen Clark. What happens next? event prediction using a compositional neural network model. In *AAAI*, volume 30, 2016.

William L Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *NeurIPSS*, pages 1025–1035, 2017.

Jiaxin Huang, Yiqing Xie, Yu Meng, Jiaming Shen, Yunyi Zhang, and Jiawei Han. Guiding corpus-based set expansion by auxiliary sets generation and co-expansion. In *WWW*, pages 2188–2198, 2020.

Xiaomeng Jin, Manling Li, and Heng Ji. Event schema induction with double graph autoencoders. In *Proc. The 2022 Conference of the North American Chapter of the Association for Computational Linguistics - Human Language Technologies (NAACL-HLT2022)*, 2022.

Seyed Mehran Kazemi and David Poole. Simple embedding for link prediction in knowledge graphs. *NeurIPS*, 31, 2018.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *ICLR*, 2015.

Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *ICLR*, 2017.

Kai Lei, Meng Qin, Bo Bai, Gong Zhang, and Min Yang. Gcn-gan: A non-linear temporal link prediction model for weighted dynamic networks. In *INFOCOM*, pages 388–396. IEEE, 2019.

Manling Li, Qi Zeng, Ying Lin, Kyunghyun Cho, Heng Ji, Jonathan May, Nathanael Chambers, and Clare Voss. Connecting the dots: Event graph schema induction with path language modeling. In *EMNLP*, pages 684–695, 2020.

Manling Li, Sha Li, Zhenhailong Wang, Lifu Huang, Kyunghyun Cho, Heng Ji, Jiawei Han, and Clare Voss. The future is not one-dimensional: Complex event schema induction by graph modeling for event prediction. In *EMNLP*, pages 5203–5215, 2021a.

Sha Li, Heng Ji, and Jiawei Han. Document-level event argument extraction by conditional generation. In *Proc. The 2021 Conference of the North American Chapter of the Association for Computational Linguistics - Human Language Technologies (NAACL-HLT2021)*, 2021b.

Ying Lin, Heng Ji, Fei Huang, and Lingfei Wu. A joint neural model for information extraction with global features. In *Proc. The 58th Annual Meeting of the Association for Computational Linguistics (ACL2020)*, 2020.

Qing Lyu, Li Zhang, and Chris Callison-Burch. Reasoning about goals, steps, and temporal ordering with wikihow. *EMNLP*, 2020.

Piyush Mishra, Akanksha Malhotra, Susan Windisch Brown, Martha Palmer, and Ghazaleh Kazeminejad. A graphical interface for curating schemas. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing: System Demonstrations*, pages 159–166, 2021.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *NeurIPS*, 32, 2019.

Ali Sadeghian, Mohammadreza Armandpour, Patrick Ding, and Daisy Zhe Wang. Drum: End-to-end differentiable rule mining on knowledge graphs. In *NeurIPS*, volume 32, 2019.

Jiaming Shen, Zeqiu Wu, Dongming Lei, Jingbo Shang, Xiang Ren, and Jiawei Han. Setexpan: Corpus-based set expansion via context feature selection and rank ensemble. In *ECML-PKDD*, pages 288–304. Springer, 2017.

Zhiqing Sun, Zhi-Hong Deng, Jian-Yun Nie, and Jian Tang. Rotate: Knowledge graph embedding by relational rotation in complex space. In *ICLR*, 2018.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. In *ICLR*, 2018.

Hongwei Wang, Fuzheng Zhang, Min Hou, Xing Xie, Minyi Guo, and Qi Liu. Shine: Signed heterogeneous information network embedding for sentiment link prediction. In *WSDM*, pages 592–600, 2018.

Hongwei Wang, Hongyu Ren, and Jure Leskovec. Relational message passing for knowledge graph completion. In *KDD*, pages 1697–1707, 2021.

Lilian DA Wanzare, Alessandra Zarcone, Stefan Thater, and Manfred Pinkal. A crowdsourced database of event sequence descriptions for the acquisition of high-quality script knowledge. 2016.

Noah Weber, Niranjan Balasubramanian, and Nathanael Chambers. Event representations with tensor-based compositions. In *AAAI*, volume 32, 2018.

Noah Weber, Rachel Rudinger, and Benjamin Van Durme. Causal inference of script knowledge. In *EMNLP*, pages 7583–7596, 2020.

Haoyang Wen, Ying Lin, Tuan Lai, Xiaoman Pan, Sha Li, Xudong Lin, Ben Zhou, Manling Li, Haoyu Wang, Hongming Zhang, et al. Resin: A dockerized schema-guided cross-document cross-lingual cross-media information extraction and event tracking system. In *NAACL-HLT*, pages 133–143, 2021.

Bishan Yang, Scott Wen-tau Yih, Xiaodong He, Jianfeng Gao, and Li Deng. Embedding entities and relations for learning and inference in knowledge bases. In *ICLR*, 2015.

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

Yue Yang, Artemis Panagopoulou, QING LYU, Li Zhang, Mark Yatskar, and Chris Callison-Burch. Visual goal-step inference using wikihow. In *EMNLP*, 2021.

Muhan Zhang and Yixin Chen. Link prediction based on graph neural networks. *NeurIPS*, 31:5165–5175, 2018.

Shuai Zhang, Yi Tay, Lina Yao, and Qi Liu. Quaternion knowledge graph embeddings. In *NeurIPS*, pages 2735–2745, 2019.

Xiyang Zhang, Muhao Chen, and Jonathan May. Salience-aware event chain modeling for narrative understanding. In *EMNLP*, 2021.

Shuyan Zhou, Li Zhang, Yue Yang, QING LYU, Pengcheng Yin, Chris Callison-Burch, and Graham Neubig. Show me more details: Discovering hierarchies of procedures from semi-structured web data. In *ACL*, 2022.

Appendix

A Limitation of Integer Programming Based Graph Matching Algorithm

For an instance event graph I and schema graph S, 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:

$$\begin{aligned} &\max_{\mathbf{X}} \sum_{\mathbf{X}_{ij}=1} s_N(i,j) + \sum_{\substack{\mathbf{X}_{ik}=1 \\ \mathbf{X}_{jl}=1}} s_L(\langle i,j\rangle, \langle k,l\rangle) \\ &s.t. \begin{cases} \mathbf{X} \in \{0,1\}^{|I|\times|S|} \\ \sum_{j=1}^{|S|} \mathbf{X}_{ij} \leq 1, \text{ for } i = 1, \cdots, |I|, \end{cases} \end{aligned} \tag{10}$$

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. (10) defines a 0-1 integer programming problem [Cho et al., 2014], which is NP-complete. To improve 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.

Based on the above two facts, we propose the two-stage heuristic graph matching algorithm, which greatly reduces the time overhead of integer programming based graph matching algorithm. Note that in this two-stage matching scheme, the first stage (Eq. (1)) and the second stage (Eq. (2)) exactly correspond to the two terms in the objective function in Eq. (10), i.e., node similarity and link similarity.

B Computational Complexity

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. (1) is $O(N_S)$, and the complexity of calculating $\mathcal{P}_I(e_i)$, $\mathcal{F}_I(e_i)$, and $\mathcal{A}_I(e_i)$ is d. In the worst case where $|E_i| = N_S$, we need to calculate $\mathcal{P}_I(e_j)$, $\mathcal{F}_I(e_j)$, $\mathcal{A}_I(e_j)$, as well as the Jaccard index for every node $e_j \in S$, 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.

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 $\quad c' \leftarrow \arg\max_{e \in C:\ dist(e,I')=1} f(e, I')$;
6 $\quad$ **if** $f(c', I') > Threshold$ **then**
7 $\quad\quad I' \leftarrow I' \cup \{c'\}$, $C \leftarrow C \backslash c'$;
8 $\quad\quad$ Add a new event node c to $\hat{I}$ whose type is the same as c';
9 $\quad\quad$ **for** n' *in* c'*'s neighbor events and neighbor entities in* S **do**
10 $\quad\quad\quad$ **if** n' *has a matching node* n *in* $\hat{I}$ **then**
11 $\quad\quad\quad\quad$ Add a temporal or argument link between c and n in $\hat{I}$ according to the link between c' and n' in S;
12 $\quad$ **else**
13 $\quad\quad$ **break**
14 **return** $\hat{I}$

C 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* [Shen et al., 2017, Huang et al., 2020], 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 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.

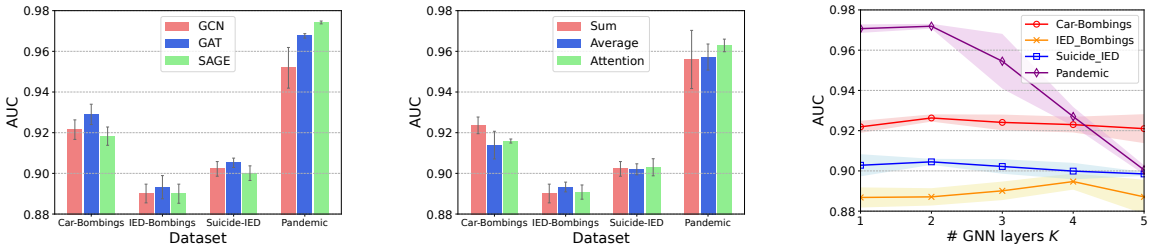
It is worth noting that missing events are predicted according to the ascending order of their distance to the subgraph I', which ensures that the original instance graph I is expanded over the schema in a natural, from-the-inside-out manner. In addition, note that the predicted missing event at one 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.

D 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 [Paszke et al., 2019]. We use GCN [Kipf and Welling, 2017] 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 [Kingma and Ba, 2015] 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 E. 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: $\{32, 64, 128, 256, 512\}$;



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

Figure 4: Hyperparameter sensitivity of SEGC-Neighbor w.r.t. GNN type, READOUT function, and the number of GNN layers K.

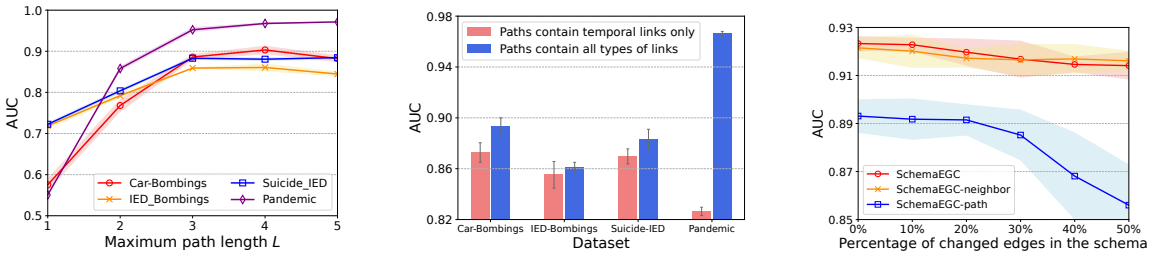
- READOUT function: {sum, average, attention};
- The maximum path length: $\{1, 2, 3, 4, 5\}$;
- The dimension of hidden layer in $\text{MLP}_{neighbor}$ and MLP_{path}: $\{32, 64, 128, 256, 512\}$;
- Batch size: $\{32, 64, 128, 256, 512\}$;
- Learning rate: $\{0.0005, 0.001, 0.005, 0.01, 0.05, 0.1\}$.

E 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 SEGC, we study the impact of GNN type, READOUT function, and the number of GNN layers K on the model performance. As shown in Figure 4a, we use three GNNs in the neighbor module: GCN [Kipf and Welling, 2017], GAT [Veličković et al., 2018], and SAGE [Hamilton et al., 2017]. 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 4b demonstrates that attention is a good choice for the READOUT function, but it does not perform universally the best on all datasets. Figure 4c 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 5a 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 5b demonstrates that considering all the three types of links for the path module works better than only considering event-event temporal links.



(a) Maximum path length L (b) Path type (c) Impact of schema quality

Figure 5: Hyperparameter sensitivity of SEGC-Path w.r.t. (a) the maximum path length L and (b) path type. (c) Impact of schema quality on Car-Bombings dataset.

F 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 5c. The performance of the path module (SEGC-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 (SEGC-neighbor) is more resistant to the noise in schema graph. As a result, the whole SEGC model is able to basically maintain its performance even when half of the edges in the schema graph are changed.