GCP: Graph Encoder with Content-Planning for Sentence Generation from Knowledge Base

Bayu Distiawan Trisedya, Jianzhong Qi, Wei Wang, and Rui Zhang*

Abstract—A knowledge base is a large repository of facts that are mainly represented as triples, each of which consists of a subject, a predicate, and an object. The triples together form a graph, i.e., a knowledge graph. The triple representation in a knowledge graph offers a simple interface for applications to access the facts. However, this representation is not in a natural language form, which is difficult for humans to understand. We address this problem by proposing a system to translate a set of triples (i.e., a graph) into natural sentences. We take an encoder-decoder based approach. Specifically, we propose a ***G**raph encoder with **C**ontent-**P**lanning capability* (**GCP**) to encode an input graph. GCP not only works as an encoder but also serves as a content-planner by using an entity-order aware topological traversal to encode a graph. This way, GCP can capture the relationships between entities in a knowledge graph as well as providing information regarding the proper entity order for the decoder. Hence, the decoder can generate sentences with a proper entity mention ordering. Experimental results show that GCP achieves improvements over state-of-the-art models by up to 3.6%, 4.1%, and 3.8% in three common metrics BLEU, METEOR, and TER, respectively.

Index Terms—Natural language processing, triple-to-text generation, knowledge base.

1 Introduction

Knowledge bases in the form of *Knowledge graphs* (KGs) are becoming an enabling resource for many applications, including question answering systems, recommender systems, and summarization tools (for more details see survey [1], [2]). A fact in a KG is stored as a triple that consists of three elements in the form of ⟨`subject, predicate, object`⟩. It describes a relationship between an entity (the subject) and another entity or literal (the object) via the predicate. This representation allows easy data share between KGs. However, the elements of a triple are typically stored as *Uniform Resource Identifiers*, and many predicates (words or phrases) are not intuitive; this representation is difficult to comprehend by humans.

In this paper, we study *text generation* [3] from structured data. Specifically, we aim to translate a set of triples (i.e., a graph) into natural sentences to help humans comprehend the knowledge embedded in the triples. We call this task *triple-to-text generation*. This task has many applications, such as description generation [4] and question answering [5], [6]. For example, in description generation, the goal is to generate a sentence that describes a target entity in a knowledge base. Table 1 illustrates such an example. Suppose we are interested in an entity `David Cameron`. First, a description generation system extracts all information about `David Cameron` by querying a knowledge base and retrieves three related triples ⟨`David Cameron, birth place, London`⟩, ⟨`David Cameron, birth date, 1966-10-09`⟩, and ⟨`London ,capital of, England`⟩. Then, the system generates a natural sentence that incorporates the information of the triples to describe the entity. In this example, the generated description is "`David Cameron was born on October 9, 1966 in London, the capital of England`".

TABLE 1
Data-to-text generation from knowledge base triples.

Triples	⟨`David Cameron, birth place, London`⟩ ⟨`David Cameron, birth date, 1966-10-09`⟩ ⟨`London, capital of, England`⟩
Target Sentence	`David Cameron was born on October 9, 1966 in London, the capital of England.`

Recent neural triple-to-text models use the encoder-decoder framework [7], [8]. They propose graph encoders to exploit the input structure. A graph encoder for sentence generation requires to capture the relationships between entities in a triple (*intra-triple relationships*, e.g., the relationship between `David Cameron` and `London`) and the relationships between entities in related triples (*inter-triple relationships*, e.g., a multi-hop relationship `birth_country` between `David Cameron` and `England`, which may be captured via entity `London`). Vougioklis et al. [9] propose a graph encoder using feed-forward neural networks. This encoder captures intra-triple relationships but may fail to capture inter-triple relationships. Marcheggiani and Perez-Beltrachini [10] propose a *GCN encoder* that employs Graph Convolutional Networks [11] to encode the input graph. The GCN encoder captures the relationship between an entity and its close neighbors but may fail to capture multi-hop relationships between entities (i.e., multi-hop inter-triple relationships). Koncel-Kedziorski et al. [12] propose *GraphWriter* combining Graph Attention Networks (GAT) [13] and Transformer [14] as an encoder to handle multi-hop inter-triple relationships.

In text generation, especially in triple-to-text generation, there are two main challenges. The first is selecting important data (entities) from a set of triples to be mentioned

- *B. D. Trisedya is with Universitas Indonesia and The University of Melbourne.*
  E-mail: bayu.trisedya@unimelb.edu.au
- *J. Qi is with The University of Melbourne.*
  E-mail: jianzhong.qi@unimelb.edu.au
- *R. Zhang (www.ruizhang.info)*
  *E-mail: rayteam@yeah.net *R. Zhang is the corresponding author.*
- *W. Wang is with The Hong Kong University of Science and Technology.*
  E-mail: weiwcs@ust.hk

Manuscript received XXXXXX XX, XXXX; revised XXXXXX XX, XXXX.

in the target sentence (i.e., content-selection). The second is ordering the selected data to generate a concise and easy-to-understand sentence (i.e., content-planning). A fixed input entity processing order induced by our entity-aware embeddings helps reduce the diversity of the content-plan learning space, which leads to generated texts of higher quality. They all use an encoder-decoder framework without explicit content-planning, which has been shown to suffer in the coherence of the output compared with traditional pipeline methods that perform explicit content-planning (e.g., by using manually created content templates) [15]. Moreover, the encoders in these models are graph encoders, such as GCN or GAT, which just consecutively aggregate all entities in the input triples following the edges in the graph formed by the triples. This aggregation ignores the entity mention order in the target sentence and hence is sub-optimal in capturing entities' relationships for text generation. Meanwhile, adding content-planning to neural text generation models explicitly is non-trivial. An existing model [16] takes a two-stage approach and learns content-planning separately from the text generation stage. This two-stage architecture is prone to error propagation between the content-planner and the text generator. Moreover, this approach requires significant human efforts to create triples–content-plan data pairs as training data for the content-planner.

We address the above limitation of neural triple-to-text generation models by performing content-planning during the encoding process. We propose ***G**raph encoder with **C**ontent-**P**lanning capability* (**GCP**) that integrates graph encoding and content-planning capabilities in a single model. It thus avoids the error propagation problem of the aforementioned models. Since no triple-to-text generation data provides ground truth for supervised content-plan learning, we propose an entity-order aware topological traversal algorithm to encode a graph input following a proper entity order (i.e., the order of entity occurrences in natural sentences), which resembles a content-planning mechanism. Our topological traversal algorithm relies on entity-order aware embeddings to encode the content-plan.

To learn entity-order aware embeddings, we train a KG embedding model over a word-entity graph. The word-entity graph is a KG containing triples from which sentences are to be generated, enriched by *entity-word co-occurrence triples* and *entity-order triples* extracted from a text corpus in a general domain, such as Wikipedia, to allow the learn embeddings to capture the common entity order in natural sentences. Here, an entity-word co-occurrence triple is formed by word-pairs that co-occur within a window in a sentence. An entity-order triple represents a *preceding* relationship between two entities. The learned embeddings capture not only the relationships between words and entities but also the common entity mentioning order in a sentence. Hence, we can exploit the embeddings to provide supervision signals for the traversal algorithm and to initialize the proposed text generation model. We learn the pretrained entity-order aware embeddings in an unsupervised manner. Therefore, we do not require triples–content-plan data pairs as additional training data.

To capture the relationships between entities, we propose a Graph-LSTM unit that aggregates both an entity and its relationships, as opposed to the standard LSTM unit [17] that can only take one input. The combination of the entity-order aware topological traversal and the Graph-LSTM units helps the proposed GCP aggregate the hidden states of entities in a graph with a more natural order, i.e., following common entity mentioning order in a sentence. Unlike our proposed model, existing graph encoders such as the GCN encoder [10] and GraphWriter [12] do not consider such ordering. They use a simple aggregating mechanism, i.e., by averaging the hidden state of the surrounding entities. Thus, our proposed GCP provides a better graph representation than the existing graph encoders and helps the decoder generate a more natural sentence.

The main contributions of this paper are as follows.

- **C1**: We propose a text generation model with a graph encoder powered by content-planning capabilities to generate sentences from a graph (i.e., a set of triples).
- **C2**: We propose a Graph-LSTM unit that aggregates both an entity and its relationships in a single unit to capture the intra-triple and inter-triple relationships between entities in a KG.
- **C3**: We propose an entity-order aware topological traversal that aggregates the hidden states of entities in a graph in a more natural order to integrate content-planning capabilities in a graph encoder. The traversal algorithm takes supervision signals based on the order of entity mentions in natural sentences.
- **C4**: We present an entity-order aware graph embedding model trained over a knowledge graph enriched by entity-order relationships to capture the order of entities in natural sentences.
- **C5**: We evaluate the proposed model over two real datasets. The results show that our proposed model outperforms the state-of-the-art models [9], [10] consistently.

This paper is an extension of our previous conference paper [18]. In the conference paper, we propose *GTR-LSTM*, a graph encoder with a Graph-LSTM unit to capture multiple relationships between entities (i.e., contribution **C2**). To capture multi-hop relationships between entities, GTR-LSTM uses a topological traversal with a random tie-breaker, which may improperly aggregate the entities' hidden state due to its randomness. In this journal extension, we extend GTR-LSTM substantially by integrating a content-planning mechanism into the encoder (**C1**). We propose an entity-order aware topological traversal that controls the order of the hidden state computation of entities in a graph (**C3**, detailed in Section 3.4). The aggregation of the hidden states follows the common entity mentioning order in a sentence. Hence, our proposed encoder provides a better graph representation that is aware of the proper entity ordering to help the decoder generate a sentence. We propose an entity-order aware embedding model to capture the common order of entity mentions in a sentence. The learned embeddings help the encoder to encode the input graph in more natural order (**C4**, detailed in Section 3.3). We also conducted a more comprehensive experimental study, including the comparison with the state-of-the-art triple-to-text generation models that use Graph Convolutional Networks, Graph Attention Networks, and Transformer models (**C5**, detailed in Section 4).

2 Related Work

In recent years, researchers have done lots of research on KG, such as KG embedding [19], entity alignment [20], relation extraction [21], etc. This work addresses the generation of sentence to describe triples in KG. In this section, we discuss traditional and neural text generation models, including the work in triple-to-text generation.

2.1 Traditional Text Generation Models

Traditional text generation models [3] consist of three steps: (1) content-selection, which selects the data to be expressed, (2) content-planning, which decides the structure of the sentences to be generated, and (3) surface-realization, which generates the final output based on the content-planning. Most traditional models employ handcrafted rules or a shallow statistical model for content-planning [22], sentence planning [23], and surface realization [24].

In triple-to-text generation, Bontcheva and Wilks [25] follow a traditional approach to generate sentences from knowledge base triples in the medical domain. They start with filtering repetitive triples (i.e., content-selection) and then group the coherent triples (i.e., content-planning). The generated sentences of the coherent triples are aggregated to produce the final sentences (i.e., surface realization). Cimiano et al. [26] generate cooking recipes from semantic web data. They focus on using a large corpus to extract lexicon in the cooking domain. The lexicon is then used with a traditional text generation approach to generate cooking recipes. Duma and Klein [27] learn a sentence template from a parallel triples-text corpus. They first align entities in the triples with entities mentioned in sentences. Then, they extract templates from the aligned sentences by replacing the entity mention with a unique word. These methods employ rules that work well on triples in a seen domain but fail on triples in a previously unseen domain.

2.2 Neural Text Generation Models

Recently, neural text generation models are proposed. Lebret et al. [4] generate the first sentence of a biography using a conditional neural language model. This model is trained to predict the next word of a sentence based on the previously generated words and additional features captured from Wikipedia infoboxes. Liu et al. [28] propose a dual attention model as a follow-up on biography summarization. These studies employ the encoder-decoder framework [7] widely used in machine translation. They run experiments on cross-domain datasets, which show that the neural approach is more flexible for an open domain since it is not limited to handcrafted rules.

Recent studies show that adding a content-plan (i.e., the order of data to be mentioned in a sentence) to neural text generation models can improve the quality of the generated text. Sha et al. [29] propose a link-based attention model to learn the order of entity mentions in a sentence. Puduppully et al. [16] employ pointer networks [30] to select salient data and learn its order as a content-plan. Trisedya et al. [31] propose a content-plan attention model to guide the decoder for sentence generation. These models separate content-planning and text generating in two stages, which are prone to error propagation. In contrast, our proposed model integrates content-planning into the encoder to avoid error propagation.

2.3 Neural Triple-to-text Generation

Current triple-to-text generation models also employ the encoder-decoder framework. Recent studies proposed different types of encoders to encode the input triples, as opposed to an LSTM encoder that is used in the typical text generation models. Vougiouklis et al. [9] develop Neural Wikipedian, which generates a summary from the triples. It uses feed-forward neural networks to encode each triple and concatenate them as the input of the decoder. Marcheggiani and Perez-Beltrachini [10] employ graph convolutional networks (GCN) [11] as the encoder. However, these models may fail to capture inter-triple relationships. Neural Wikipedian applies feed-forward neural networks for each triple, which makes it only optimized for intra-triple relationships. Meanwhile, GCN may fail to capture long multi-hop relationships between entities because it captures multi-hop relationships by stacking GCN layers, which weakly aggregates these relationships.

The state-of-the-art in triple-to-text generation is GraphWriter [12]. It integrates Graph Attention Networks (GAT) [13] and the Transformer model [14]. The GAT captures the intra-triple and inter-triple relationships, while the self-attention mechanism in the Transformer model captures multi-hop relationships between entities in a graph.

The triple-to-text generation models above only focus on capturing the relationships between entities entailed in the given knowledge graph. They rely on the encoder-decoder model to learn entity orders implicitly. In this paper, we propose a graph encoder that performs not only encoding but also content-planning to obtain more natural entity order in the generated sentences.

2.4 Joint Learning of Word and Entity Embeddings

Joint learning of word and entity embeddings has become an essential component in many downstream applications, such as entity disambiguation, KG completion, and relation extraction. Sharing representation is the most common technique for the joint learning [32], [33]. This technique first computes the word and entity embeddings separately. It then exploits anchor text (i.e., an entity mention in a text that refers to an entity in a KG) to map both embeddings into the same vector space. Another technique exploits entity descriptions [34]. This technique forces an entity embedding to be similar to the embedding of the corresponding entity description. Since KGs may not have descriptions for all entities, this technique may be limited by training data availability. Cao et al. [35] propose a Multi-Prototype Entity Mention Embedding model, which learns multiple embeddings for an entity mention, considering that an entity mention can refer to more than one entity, e.g., `Apple` may refer to a company or a fruit. The resultant embeddings may be difficult to be used as pre-trained embeddings since a word may have multiple embeddings.

In this paper, we use joint learning of word and entity embeddings to compute the entity-order aware embeddings. We use a translation-based graph embedding model over a word-entity graph instead of entity mentions or descriptions, which differs from all existing joint learning methods (detailed in Section 3.3).

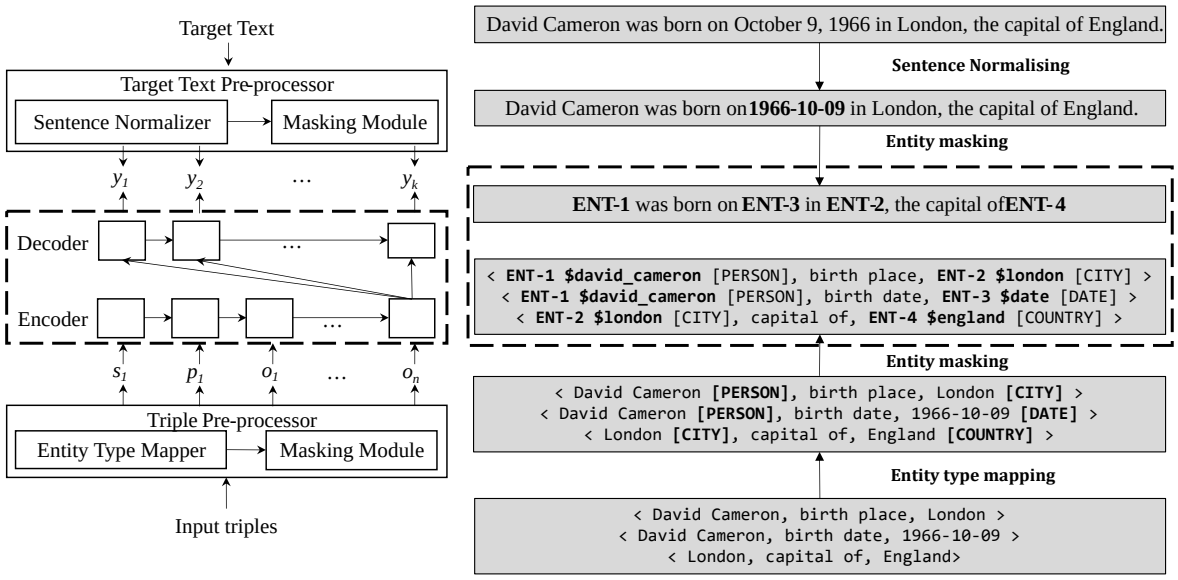


Fig. 1. Data-to-text sentence generation from knowledge base triples based on an encoder-decoder architecture.

3 Proposed Model

We start with the problem definition. We consider a set of triples as the input, which is denoted by $\mathcal{T} = \{t_1, t_2, ..., t_n\}$ where a triple t_n consists of three elements (subject s_n, predicate p_n, and object o_n), $t_n = \langle s_n, p_n, o_n \rangle$. Every element can contain multiple words. We aim to generate a sentence that consists of a sequence of words $\mathcal{Y} = \langle y_1, y_2, ..., y_k \rangle$, such that the relationships in the input triples are correctly represented in $\mathcal{Y}$ while the sentences have a high quality. Table 1 illustrates the input and output of the problem.

3.1 Solution Framework

Our solution framework uses an encoder-decoder architecture [7], as illustrated in Fig. 1. The framework consists of three components: a *triple pre-processor*, a *target text pre-processor*, and an *encoder-decoder module*.

The triple pre-processor consists of an entity type mapper and a masking module. The entity type mapper maps the subjects and objects in the triples to their types, such that the target sentences are learned based on entity types rather than entities. For example, the input entities in Table 1, `David Cameron`, `London`, `England`, and `1966-10-09` are mapped to `PERSON`, `CITY`, `COUNTRY`, and `DATE`, respectively. The mapping aims to improve the model generalizability in handling entities unseen in training but with known types, hence improving the output quality. The masking module converts each entity into an *entity identifier* (eid). The target text pre-processor consists of a text normalizer and a masking module. The text normalizer converts abbreviations and dates into the same format as the corresponding entities in the triples. Similar to that of the triple pre-processor, the masking module of the text processor replaces entities in the target sentences by their eids. These modules are detailed in Section 3.2.

The main component of the framework is the encoder-decoder module. In the encoder side, we propose *GCP*. Unlike the standard encoder-decoder framework, GCP is designed to accommodate KG triples as input. GCP captures both intra-triple and inter-triple entity relationships by handling cycles in the input graph and capturing multiple relationships between entities. Besides these capabilities, our proposed encoder also has the capability as a content-planner. This capability helps our model to generate a sentence with a better ordering of entity mentions.

To handle multiple relationships between entities, we propose a Graph-LSTM unit that aggregates both an entity and its relationships. Meanwhile, the content-planning capability is obtained through an entity-order aware topological traversal that controls the aggregation of the hidden states of the entity in a graph. The traversal takes supervision signals on the processing order from an *entity-order aware embeddings*. To learn such entity-order aware embeddings, we train a translation-based graph embedding model over a word-entity graph. The word-entity graph is a KG that contains triples from which sentences are to be generated and has been enriched by entity-word co-occurrence triples and entity-order triples extracted from a text corpus, such as Wikipedia. Here, an entity-word co-occurrence triple is formed by word-pairs that co-occur within a window in a sentence. An entity-order triple represents a *preceding* relationship between two entities. The entity-order aware embedding models and the proposed GCP are detailed in Section 3.3 and Section 3.4, respectively.

3.2 Entity Masking

Entity masking makes our proposed framework generalize better to entities unseen in training. This technique addresses the problem of limited training data faced by many natural language generation tasks.

Entity masking replaces entity mentions with eids, gids, and $type$s in both the input triples and the target sentences. Here, $type$ represents an entity type, and eid represents a local identifier to differentiate entities in a given input. They together help learn the output positions of entities base on their types without knowing the actual entity names. Meanwhile, gid is a global entity identifier to differentiate entities in the entire dataset. It helps learn entity-specific mapping from input triples to output position and context. In the encoder side, the subject s_n and the object o_n of a triple $t_n = \langle s_n, p_n, o_n \rangle$ are transformed into $\langle eid, gid, type \rangle$. The

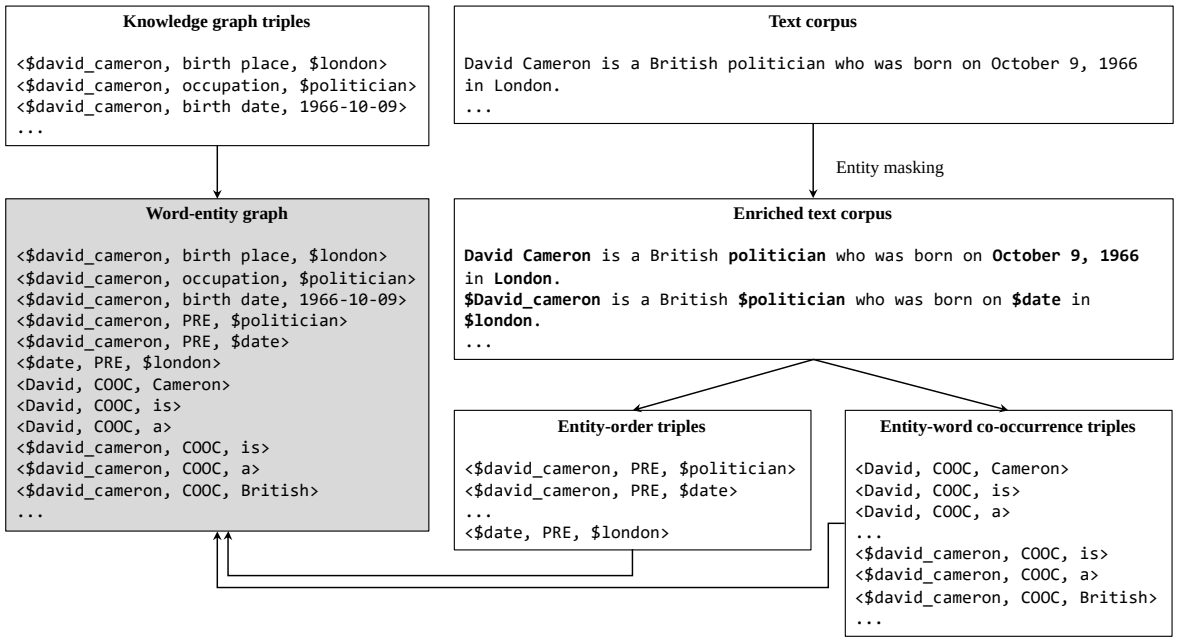


Fig. 2. The construction of a word-entity graph.

predicate p_n is preserved since it indicates the relationship between the subject and the object. In the decoder side, the entities in the target text are replaced by their corresponding eids. As illustrates in Fig. 1, the entity `David Cameron` is replaced by `"ENT-1 $david_cameron [PERSON]"`[1] in the input triple and `"ENT-1"` in the target sentence. Here, `ENT-1`, `$david_cameron`, and `[PERSON]` are the local identifier, global identifier, and entity type, respectively.

To obtain the entity type, the entity type mapper uses DBpedia [36] lookup API. Since the datasets used in the experiments are extracted from Wikipedia, the API provides type information for all entities in the datasets. The API may return multiple types for an entity. The type assigned for each entity is determined by its level in the WordNet [37] hierarchy. We take the type with the highest level (i.e., the most specific level) in the hierarchy.

The local identifiers are generated locally for each training sample. We assign a unique eid to each entity in the input triples and match the entity with the entity mentions in the target sentence. Entity matching to generate the eids is not the focus of our study. We use three matching methods to find entity mentions in the target sentence: exact matching, *n-gram* matching, and parse tree matching. Exact matching is used to find the exact mentions; n-gram matching is used to handle partial matching with the same number of words (e.g., `"David Cameron"` and `"David C."`); and parse tree matching is used to find partial matching with a different number of words (e.g., `"David Cameron"` and `"David W. Cameron"`).

The global identifiers are generated uniquely to differentiate entities in the entire dataset. For unseen entities in testing, the gid is set to a unique identifier `$unk`. In such a scenario, the entity types play critical roles to differentiate entities in the input.

1. To make the following examples more intuitive, we use the first word of the entity mention to represent the entity instead of using eid and gid. For example, we use `"David"` instead of `"ENT-1 $david_cameron"` in Fig. 3.

In training data preparation, before masking the entities in the target sentence, we normalize the target sentence to convert abbreviations and dates into the same format as the corresponding entities in the input triples. To convert abbreviations, we use a dictionary of acronyms extracted from Wikipedia [38]. To convert dates, we use regular expressions. Fig. 1 illustrates the sentence normalization procedure. The string `"October 9, 1966"` is replaced by `"1966-10-09"` to match the date format on the triple.

3.3 Entity-order Aware Embedding Model

Since we aim to generate sentences based on a given set of triples, it is critical to learn embeddings that preserve the relationships between the words that form the sentences and the entities that form the triples. A joint embedding learning model can help capture the relationship between words and entities as well as the relationship between words and entities themselves. To train such an embedding model, we follow Wang et al. [32] and Yamada et al. [33] and take a translation-based graph embedding approach. Such an approach (e.g., TransE and variants [19], [39]–[41]) considers that the embedding of the object o of a triple should be close to the embedding of the subject s plus the embedding of the predicate p, i.e., $\mathbf{s} + \mathbf{p} \approx \mathbf{o}$. This approach preserves entity structure information, i.e., entities that share similar neighbors in a KG should have a close representation in the embedding space.

To capture relationships between words and entities, we train a translation-based graph embedding model over a word-entity graph constructed as follows (cf. Fig. 2). The core of the word-entity graph is a KG containing triples from which sentences are to be generated. This can be a general domain knowledge base, such as DBpedia [36].

We add two sets of triples to the core graph. The first contains entity-word co-occurrence triples, and the second contains entity-order triples. Both sets of triples are extracted from a text corpus such as Wikipedia. First, we apply entity masking (cf. Section 3.2) to the text corpus to enrich

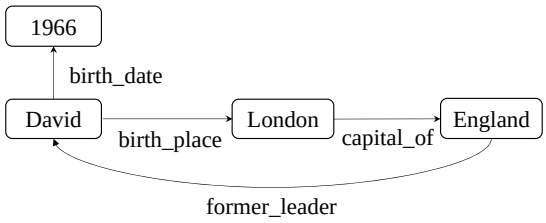


Fig. 3. A small knowledge graph formed by a set of triples.

them with masked sentences, where entity mentions are replaced by their global identifiers. The masked sentences not only capture relationships between words and entities but also help in entity disambiguation [33].

To collect entity-word co-occurrence triples, we extract word pairs that co-occur in the enriched text corpus within a window $\mathcal{W}_1$ ($\mathcal{W}_1 = 5$ in our experiments) and a frequency of co-occurring for at least 5 times. For each extracted pair $\langle w_1, w_2 \rangle$, we create a triple by adding a predefined relationship `COOC`. For example, $\langle$`David,COOC,Cameron`$\rangle$[2] is extracted from the raw sentence, and $\langle$`$david_cameron,COOC,is`$\rangle$[3] is extracted from the corresponding masked sentence. These triples are added to the word-entity graph in Fig. 2.

To collect entity-order triples, first, we extract a list of gids from left to right in the masked sentences, e.g., [`$david_cameron, $politician, $date, $london`]. Then, we extract the entity pairs using a sliding window $\mathcal{W}_2 = 2$ from the list. For each extracted pair $\langle e_1, e_2 \rangle$, we create a triple by adding a predefined relationship `PRE`. For example, $\langle$`$david_cameron,PRE,$politician`$\rangle$ is added to the word-entity graph in Fig. 2. This triple represents a preceding relationship, i.e., `$david_cameron` is mentioned before `$politician` in a sentence.

Based on the word-entity graph, our embedding model learns the entity and word embeddings by minimizing a margin-based objective function:

$$\mathcal{J}_E = \sum_{t_r \in \mathcal{T}_r} \sum_{t'_r \in \mathcal{T}'_r} \max\left(0, \left[\gamma + f(t_r) - f(t'_r)\right]\right) \quad (1)$$

$$\mathcal{T}_r = \{\langle s, p, o \rangle | \langle s, p, o \rangle \in \mathcal{G}_{WE}\} \quad (2)$$

$$\mathcal{T}_r{}' = \left\{\langle s', p, o \rangle \mid s' \in \mathcal{Z}\right\} \cup \left\{\langle s, p, o' \rangle \mid o' \in \mathcal{Z}\right\} \quad (3)$$

$$f(t_r) = \|\mathbf{M_p s} + \mathbf{p} - \mathbf{M_p o}\|_2 \quad (4)$$

Here, $\|\cdot\|_2$ is the L2-Norm, γ is a margin hyperparameter, $\mathcal{T}_r$ is the set of valid triples from a word-entity graph $\mathcal{G}_{WE}$, $\mathcal{T}'_r$ is the set of corrupted triples, and $\mathcal{Z}$ is the set of entities (subject s and object o of a triple) in $\mathcal{G}_{WE}$. The corrupted triples are used as negative samples created by replacing the subject or object of a valid triple in $\mathcal{T}_r$ with a random entity. Since we extract triples from text corpus based on word co-occurrences, the word-entity graph may consist of reflexive (e.g., $\{\langle s, p, o \rangle, \langle o, p, s \rangle\}$), one-to-many (e.g., $\{\langle s, p, o_1 \rangle, \langle s, p, o_2 \rangle\}$), many-to-one (e.g., $\{\langle s_1, p, o \rangle, \langle s_2, p, o \rangle\}$), and many-to-many (e.g., $\{\langle s_1, p, o_1 \rangle, \langle s_1, p, o_2 \rangle, \langle s_2, p, o_1 \rangle, \langle s_2, p, o_2 \rangle\}$) relationships. To handle these types of relationships, we employ TransR [40] (cf. Eq. (4)), where the subject and object embeddings ($\mathbf{s}$ and $\mathbf{o}$) are projected into a relation space by a learned matrix $\mathbf{M_p}$ to compute a plausibility score $f(t_r)$ for each corresponding predicate p.

The advantages of our embeddings are twofold. First, our embeddings capture the relationships between words and entities. In Fig. 2, the embeddings capture the relationship between the entity `$david_cameron` and the word `British`. Second, our embeddings preserve the entity order information in a sentence that helps break ties in the topological traversal over the input graph for our encoder.

We train the embedding model before training the encoder-decoder model. We randomly initialize the embeddings and train the embedding model. Then, the learned embeddings are used to initialize the embeddings used in all the models tested, including the proposed and the baseline models (Section 4.1).

3.4 GCP

A graph encoder for sentence generation requires to capture the relationships between entities in a triple (i.e., intra-triple relationships) and the relationships between entities in related triples (i.e., inter-triple relationships). GCP addresses these requirements by a Graph-LSTM unit and an entity-order aware topological traversal. The Graph-LSTM unit captures multiple relationships between entities in a KG. Meanwhile, the entity-order aware traversal aggregates the entities' hidden state by following the common order of entity mentions in a sentence (i.e., content-plan). This makes our GCP serve as both an encoder and a content-planner.

GCP differs from existing graph encoders, such as the GCN encoder [10] and GraphWriter [12], in handling the above problems. Neither the GCN encoder nor the GraphWriter explicitly considers entity ordering when computing entity hidden states (i.e., they simply use a weighted average of the hidden state of the surrounding entities). In comparison, the content-planning capability of our GCP encoder helps it provide a better graph representation that guides the decoder to generate a more natural sentence.

3.4.1 Node and Vertex Representation

Our GCP model takes a directed graph $\mathcal{G} = \langle \mathcal{V}, \mathcal{E} \rangle$ as the input, where $\mathcal{V}$ is a set of vertices that represent entities or literals, and $\mathcal{E}$ is a set of directed edges that represent predicates. A vertex (i.e., entity) consists of a local entity identifier eid, a global identifier gid, and an entity type, while an edge (i.e., predicate) may consist of multiple words, e.g., `"birth date"`. Hence, to represent a vertex (or an edge), we use a linear transformation as follows.

$$\mathbf{x} = \tanh(\mathbf{W}[\mathbf{w_1}; \mathbf{w_2}; ...; \mathbf{w_j}] + \mathbf{b}) \quad (5)$$

where $[;]$ denotes vector concatenation, $\mathbf{b}$ denotes bias vector, $\mathbf{w}_j$ denotes the embedding of a word in a vertex (or en edge). We use zero paddings to handle different numbers of words in a vertex. $\mathbf{W}$ is a learned parameter matrix.

3.4.2 Graph-LSTM Unit

GCP computes a hidden state by taking into account the processed entity and its edge (the edge pointing to the current entity from the previous entity) to handle multiple

2. This kind of triples captures the relationship between words. They also resemble the relationship between words in the skip-gram model.

3. This kind of triples captures the relationship between entities and words, and the relationship between entities, e.g., $\langle$`$david_cameron,COOC,$london`$\rangle$.

relationships between entities in a KG. Thus, our Graph-LSTM unit (cf. Fig. 4(a)) receives two inputs: the entity and its relationship. We propose the following model to compute the hidden state of each Graph-LSTM unit.

$$\mathbf{i}_t = \sigma\left(\sum_e \left(\mathbf{U}^{ie}\mathbf{x}_{te} + \mathbf{W}^{ie}\mathbf{x}'_{t-1}\right)\right) \quad (6)$$

$$\mathbf{f}_{te} = \sigma\left(\mathbf{U}^{f}\mathbf{x}_{te} + \mathbf{W}^{f}\mathbf{x}_{t-1}\right) \quad (7)$$

$$\mathbf{o}_t = \sigma\left(\sum_e \left(\mathbf{U}^{oe}\mathbf{x}_{te} + \mathbf{W}^{oe}\mathbf{x}'_{t-1}\right)\right) \quad (8)$$

$$\mathbf{g}_t = \tanh\left(\sum_e \left(\mathbf{U}^{ge}\mathbf{x}_{te} + \mathbf{W}^{ge}\mathbf{x}'_{t-1}\right)\right) \quad (9)$$

$$\mathbf{h}_t = \left(\mathbf{h}_{t-1} * \sum_e \mathbf{f}_{te}\right) + (\mathbf{g}_t * \mathbf{i}_t) \quad (10)$$

$$\mathbf{x}'_t = \tanh(\mathbf{h}_t) * \mathbf{o}_t \quad (11)$$

Here, $\mathbf{U}$ and $\mathbf{W}$ are learned parameter matrices, σ denotes the sigmoid function, $*$ denotes element-wise multiplication, $\mathbf{x}$ is the input at the current time-step, and $\mathbf{x}'_{t-1}$ is the hidden state of the previously processed vertex. The input gate $\mathbf{i}$ determines the weight of the current input. The forget gate $\mathbf{f}$ determines the weight of the previous state. The output gate $\mathbf{o}$ determines the weight of the cell state forwarded to the next time-step. The state $\mathbf{g}$ is the candidate hidden state used to compute the internal memory unit $\mathbf{h}$ based on the current input and the previous state. The subscript t is the time-step. The subscript/superscript e is the input element (an entity or a predicate). Following Tree LSTM [42] and Graph LSTM [43], we also use a separate forget gate for each input that allows the Graph-LSTM unit to incorporate information from each input selectively.

3.4.3 Aggregating Entity Hidden State

The goals of GCP are twofold. The first is to encode a graph input to capture the relationship between entities in it. The second is to serve as a content-planner that informs the decoder regarding the proper order of entity mentions in the target sentence. To achieve these goals, GCP uses an *entity-order aware topological traversal* over the input graph to compute the hidden states of the entities. The traversal decides which hidden states to be computed next by sequentially taking a vertex with zero in-degree until all vertices are processed. Our intuition is that the order of entity mentions in a sentence follows the direction of the edges. For example, given a triple ⟨`David, birth place, London`⟩, the graph representation contains a directed edge `birth place` from `David` to `London`, and a common sentence to describe the graph is `"David was born in London"`.

Since the input graph may contain cycles (e.g., Fig. 3), there could be no vertices with zero in-degree to be visited next. In this case, we need to provide supervision signals to break ties (i.e., tie-breaking procedure) in the traversal. The supervision signal comes from the learned embeddings, which preserve the information about the order of entity mentions in a sentence (cf. Section 3.3). Given two vertices v_1 and v_2, the entity to be processed earlier is decided based on their precedence computed by their embeddings. If there is a `PRE` relationship between the entities e_1 and e_2 that correspond to v_1 and v_2, then v_1 should be processed first. Otherwise, we process v_2 first. We use a function f_{pre} to denote the computation of the vertex to be processed first:

$$f_{pre}(v_1, v_2) = \begin{cases} v_1, & \text{if } \cos(\mathbf{M_p e_1} + \mathbf{p_{pre}}, \mathbf{M_p e_2}) \geq 0 \\ v_2, & \text{otherwise} \end{cases} \quad (12)$$

where $\mathbf{e_1}$ and $\mathbf{e_2}$ are the embeddings of e_1 and e_2, respectively. $\mathbf{M_p}$ is the learned projection matrix, and $\mathbf{p_{pre}}$ is the embeddings of predicate `PRE` (i.e., a predicate that indicates the relative position between two entities), and $\cos(\mathbf{x}, \mathbf{y})$ is the cosine similarity between two vectors $\mathbf{x}$ and $\mathbf{y}$. If there are more than two vertices that have the same smallest in-degree, the function in Eq. (12) is run recursively, e.g., $f_{pre}(f_{pre}(v_1, v_2), v_3)$ for three vertices v_1, v_2, and v_3. When a vertex v_i is visited, the hidden states of all adjacent vertices of v_i are computed (or updated if it is already computed in the previous step).

Take the graph in Fig. 3 as an example. The order of hidden state computation is as follows. The process starts with a vertex with zero in-degree. Since there is no such vertex, we use Eq. (12) to determine the next vertex to be processed. Suppose that `David` is chosen as the starting vertex. Then $\mathbf{x}'_{david}$ is computed using $\mathbf{x}_0$ as the previous hidden state, and all directed edges started from `David` are removed. Next, $\mathbf{x}'_{1966}$ and $\mathbf{x}'_{london}$ are computed consecutively (vertex `1966` is selected by Eq. (12), and vertex `London` is selected in the next traversal step) by passing $\mathbf{x}'_{david}$ as the previous hidden state. Next, all directed edges started from `London` are removed (there are no directed edges started from `1966`). In the last step, $\mathbf{x}''_{david}$ is updated. Fig. 4(a) illustrates the overall process.

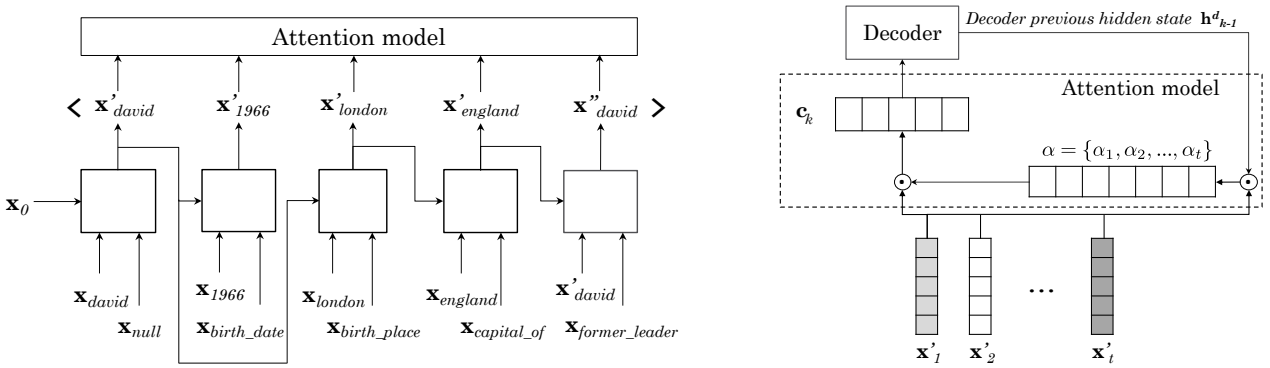
3.4.4 Capturing Global Information of the Input Graph

From Fig. 4(a), we can see that the traversal creates two branches, one ended in $\mathbf{x}'_{1966}$, and the other ended in $\mathbf{x}''_{david}$. After the encoder has computed the hidden state of each vertex, $\mathbf{x}''_{david}$ does not include the information of $\mathbf{x}'_{1966}$ and vice versa. Moreover, the graph can contain cycles that cause difficulties in determining the starting and ending vertices. Our traversal procedure ensures that the hidden states of all vertices are updated based on their adjacent vertices (local neighbors). To further capture the global information of the graph, we apply an attention model [44] on GCP. The attention model takes as input the hidden states of all vertices computed by the encoder and the previous hidden state of the decoder. It computes the context vector of the decoder in each time-step. Fig. 4(b) illustrates the attention model of GCP. We use the following equation to compute the weights of each vertex.

$$\alpha_t = \frac{\exp(\mathbf{h}^d_{k-1}{}^{\intercal}\mathbf{W}\mathbf{x}'_t)}{\sum_{j=1}^{|\mathcal{X}|}\exp(\mathbf{h}^d_{k-1}{}^{\intercal}\mathbf{W}\mathbf{x}'_j)} \quad (13)$$

Here, $\mathbf{h}^d_{k-1}$ is the previous hidden state of the decoder, $|\mathcal{X}|$ is the total number of entities (vertices) in the input triples, $\mathbf{W}$ is a learned parameter matrix, $\mathbf{x}'$ is the hidden state of a vertex, and $\alpha = \{\alpha_1, \alpha_2, ..., \alpha_t\}$ are the weights of the vertices. The context vector of the decoder for each time-step is computed as follows.

$$\mathbf{c}_k = \sum_{t=1}^{|X|} \alpha_t \mathbf{x}'_t \quad (14)$$



(a) Encoding step in GCP. (b) Attention model of GCP.

Fig. 4. The Proposed GCP Model.

The attention mechanism above differs from the standard sequence-to-sequence model [8]. In the standard sequence-to-sequence model, the attention is applied at the word level. In comparison, in GCP, the attention is applied at the entity level. Attention at the entity level is more intuitive in selecting the order of entity mentions in the target sentence.

3.5 Decoder

The decoder of the proposed framework is a standard LSTM. It is trained to generate the output sequence by predicting the next output word y_k conditioned on the hidden state $\mathbf{h}_k^d$. The current hidden state $\mathbf{h}_k^d$ is conditioned on the hidden state of the previous time-step $\mathbf{h}_{k-1}^d$, the output of the previous time-step y_{k-1}, and a context vector $\mathbf{c}_k$. The hidden state and the output of the decoder at time-step k are computed as:

$$\mathbf{h}_k^d = f(\mathbf{h}_{k-1}^d, y_{k-1}, \mathbf{c}_k) \quad (15)$$

$$\hat{y}_k = \text{softmax}(\mathbf{M}\mathbf{h}_k^d) \quad (16)$$

Here, $\hat{y}_k$ is the output probability distribution over the vocabulary, f is a single LSTM unit, and M is the hidden-to-output weight matrix. The encoder and the decoder are trained to maximize the conditional log-likelihood:

$$p(\mathcal{Y} \mid \mathcal{T}) = \log \sum_k \sum_{j=1}^{|V|} y'_{k,j} \times \log \hat{y}_{k,j} \quad (17)$$

Hence, the training objective is to minimize the negative conditional log-likelihood:

$$\mathcal{J} = -\frac{1}{D} \sum_{d=1}^{D} p(\mathcal{Y} \mid \mathcal{T}) \quad (18)$$

where $(\mathcal{Y}, \mathcal{T})$ is a pair of output word sequence and input triple set, y' is the one-hot vector representation of the target sentence over the vocabulary V, and D is the number of training samples. During the evaluation, we use a beam search with a beam size of 5 in the decoder.

To handle the out of vocabulary problem, we use entity masking as described in Section 3.2. We also perform experiments without entity masking. In this setup, we use a copy mechanism [45] that replaces the unknown token generated by the decoder by the input token with the highest attention score from the attention model (cf. Section 3.4.4).

4 Experiments

We evaluate our framework on two datasets. The first is the dataset from the WebNLG challenge [46]. We call it the *WebNLG* dataset. It contains 25,298 triples-text pairs, with 9,674 unique sets of triples. For each set, there are multiple sentences collected via crowd-sourcing as the ground truth, which may contain different entity mention orders. The tested models are trained over such a dataset to learn to generate different variations of entity mention orders. The dataset consists of a Train+Dev dataset and a Test Unseen dataset. We split Train+Dev into a training set (80%), a development set (10%), and a Seen testing set (10%) (denoted by **Seen** test dataset). The Train+Dev dataset contains triples in ten categories (topics, e.g., astronaut, monument, food, etc.), while the Test Unseen (denoted by **Unseen** test dataset) dataset has five other unseen categories. The maximum number of triples in each triple set is seven. For the second dataset, we collected data from Wikipedia pages regarding landmarks. We call it the *GKB* dataset. We first extract triples from Wikipedia infoboxes and sentences from the Wikipedia text that contain entities mentioned in the triples. Human annotators then filter out false matches to obtain $1,000$ triples-text pairs. This dataset is split into the training and development set (80%) and the testing set (20%) (denoted by **GKB** test dataset). Table 1 illustrates the data pairs of the WebNLG and GKB datasets.

We implement the proposed model using Keras and TensorFlow. We use three evaluation metrics, BLEU [47], METEOR [48], and TER [49]. For the metric computation and significance testing, we use MultEval [50].

4.1 Tested Models

We compare our model **GCP**[4] with two adaptations of a sequence-to-sequence model for triple-to-text generation and three graph encoder models.

Sequence Encoder. A straightforward adaptation of sequence-to-sequence models [7], [8] for triple-to-text generation is to transform a set of input triples $\mathcal{T}$ into a sequence of tokens, i.e., $\mathcal{T} = \{\mathbf{w}_1, \mathbf{w}_2, ..., \mathbf{w}_m\}$, where m is the number of tokens in the triples. Here, $\mathbf{w}_i$ represents the embedding of a token. This sequence forms an input for the encoder. In this adaptation, we use two types of sequence-to-sequence models, including an LSTM-based model (**S-LSTM**) and a Transformer-based model (**S-TRANS**).

Triple Encoder. The sequence encoder suffers from failing to capture both intra-triple relationships and inter-triple relationships since linearizing a set of triples may discard the graph structure. In the triple encoder adaptation, we

4. Link to code and data: https://bitbucket.org/bayudt/gtrlstm/src/master/GCP/

TABLE 2
Comparison of model performance.

Model	BLEU ↑			METEOR ↑			TER ↓		
	Seen	Unseen	GKB	Seen	Unseen	GKB	Seen	Unseen	GKB
Without entity masking									
S-LSTM	43.46±.24	23.40±.40	26.97±.23	35.25±.14	28.34±.29	26.42±.37	56.27±.33	69.43±.28	67.59±.39
T-LSTM	45.33±.39	27.31±.32	27.27±.19	34.94±.38	29.54±.44	28.80±.34	50.63±.22	61.96±.17	59.41±.25
S-TRANS	44.40±.31	24.45±.26	27.87±.15	35.18±.13	28.75±.35	27.07±.27	56.01±.17	69.30±.33	66.49±.19
T-TRANS	45.65±.12	27.53±.28	27.72±.19	34.37±.28	29.98±.19	28.53±.11	51.41±.30	61.79±.39	59.03±.23
TFF	45.29±.32	27.32±.28	26.22±.18	33.83±.36	29.07±.19	28.17±.31	52.61±.45	62.88±.26	60.53±.47
GCN	51.49±.16	29.08±.15	30.85±.13	36.82±.18	30.18±.17	29.67±.15	46.58±.13	62.24±.16	58.80±.20
GraphWriter	51.62±.21	28.14±.14	30.65±.22	36.74±.11	29.77±.21	30.02±.17	46.41±.22	62.46±.11	58.55±.15
GCP (proposed)	53.99±.16	31.07±.15	38.37±.13	37.25±.12	28.31±.17	30.65±.20	44.58±.16	59.21±.14	54.58±.11
With entity masking									
S-LSTM	49.61±.18	27.31±.15	29.36±.32	39.66±.47	28.80±.47	29.67±.17	50.07±.23	65.75±.27	65.93±.31
T-LSTM	51.84±.27	31.70±.33	32.67±.20	38.63±.13	31.26±.19	29.26±.20	46.27±.32	58.88±.32	56.26±.16
S-TRANS	49.27±.29	26.49±.18	29.35±.24	39.84±.34	29.21±.19	29.84±.16	49.35±.24	65.56±.21	66.51±.27
T-TRANS	51.51±.29	31.79±.29	32.19±.30	39.04±.22	31.39±.33	29.21±.11	46.14±.32	58.74±.31	56.97±.15
TFF	46.93±.24	27.95±.21	30.20±.26	35.67±.31	30.96±.26	28.81±.39	49.04±.16	61.22±.19	58.14±.54
GCN	56.31±.24	32.46±.15	41.25±.14	39.01±.19	30.73±.16	32.71±.12	42.41±.15	58.53±.13	51.95±.13
GraphWriter	56.45±.26	32.59±.20	42.12±.18	39.30±.20	31.32±.15	33.26±.11	42.28±.14	58.74±.11	51.52±.16
GCP (proposed)	**58.53**±.11	**36.61**±.24	**45.14**±.11	**40.93**±.21	**32.30**±.14	**35.23**±.16	**40.67**±.16	**57.08**±.15	**50.35**±.13

aggregate the elements of the same triple to retain the intra-triple relationship. The adaptation is done by first grouping the elements of each triple, i.e., the input is represented as $\mathcal{T} = \{\langle \mathbf{w}_{1,1}, ..., \mathbf{w}_{1,j} \rangle, ..., \langle \mathbf{w}_{n,1}, ...\mathbf{w}_{n,j} \rangle\}$, where $\mathbf{w}_{n,j}$ is the embedding of the j-th token in the n-th triple. Then, we apply a sequence encoder hierarchically. The first step is to apply the sequence encoder for each group (i.e., a triple) to obtain the triple representation. The second step is to apply another sequence encoder over the sequence of triple representations obtained. We use two types of sequence-to-sequence models, including an LSTM-based model (**T-LSTM**) and a Transformer-based model (**T-TRANS**).

Graph Encoder. We also compare with three existing graph encoder models, including:

- **TFF encoder** [9], which is a graph encoder using feed-forward neural networks.
- **GCN encoder** [10], which is a graph encoder based on the Graph Convolutional Networks.
- **GraphWriter** [12], which is the state-of-the-art graph encoder that combines Graph Attention Networks and Transformer. We remove the title encoder of the GraphWriter since the dataset does not include titles for each set of triples.

4.2 Hyperparameters

We use grid search to find the best hyperparameters for the models. For the embedding model, we choose the embedding dimensionality among $\{50, 75, 100, 200\}$, the learning rate of the optimizer among $\{0.001, 0.01, 0.1\}$, and the margin γ among $\{1, 5, 10\}$. We train the embedding model with a batch size of 100 and a maximum of 400 epochs. We use 512 hidden units (dimensions) for both the encoder and the decoder. We use a 0.5 dropout rate for regularization on both the encoder and the decoder to avoid over-fitting. We find that using adaptive learning rates for optimization is efficient and leads to faster converge. Thus, we use Adam [51] with a learning rate of 0.0002 instead of stochastic gradient descent.

4.3 Effect of Entity Masking

Table 2 shows the overall comparison of model performance. It shows that entity masking gives a consistent improvement for all models. Generalizing the input triples and target sentences helps the models to learn the relationships between entities from their types. This is particularly helpful when there is limited training data. We use a combination of exact matching, *n-gram* matching, and parse tree matching to find the entity mentions in the sentence. The entity masking accuracy (i.e., the precision of correctly recognizing the entities to be masked) for the WebNLG dataset is 87.15%, while for the GKB dataset is 82.45%.

Entity masking improves the BLEU score of the proposed model by 8.4% (from 53.99 when *not* using entity masking to 58.53 when using entity masking), 17.8%, and 17.6% on the **Seen**, **Unseen**, and **GKB** test datasets, respectively. Using entity masking improves not only the output quality but also the running time by requiring a smaller vocabulary.

4.4 Effect of Models

Table 2 also shows that the proposed model GCP achieves a consistent improvement over the baseline models, which is statistically significant, with $p < 0.05$ based on the t-test of all metrics. We use MultEval to compute the p-value based on approximate randomization [50]. The improvement in the BLEU score indicates that GCP reduces errors in the generated sentence. Our manual inspection confirms this result. The better (lower) TER score suggests that GCP generates a more compact output (i.e., better aggregation).

Table 3 shows a sample output of all models. The baseline models produce sentences that contain errors (e.g., the GCN output contains a wrong aggregation, "`the elizabeth tower, wembley stadium, and london is in england`"). Moreover, the baseline models generate sentences with a weak composition (e.g., `is in london` is repeated in a single sentence for GraphWriter). GCP successfully avoids these problems.

TABLE 3
Sample output of the system. The error is highlighted in bold.

Input	⟨Elizabeth Tower, location, London⟩,⟨Wembley Stadium, location, London⟩, ⟨London, capital of, England⟩,⟨England, former leader, David Cameron⟩
Reference	london , england is home to wembley stadium and the elizabeth tower. david cameron is one of the former leader of england .
S-LSTM	**england ' s leader is david cameron and place in the city of london** . the elizabeth tower is located **in england** and is **located in the wembley stadium.**
T-LSTM	the wembley stadium is located in london , england . the elizabeth tower is located in england. **one of the minister of england is the leader david cameron.**
S-TRANS	the wembley stadium is located in london **and the elizabeth tower is located in the city of london . england is lead by david cameron.**
T-TRANS	england is the location of elizabeth tower and the wembley stadium . **david cameron lives in london.**
GCN	**the elizabeth tower, wembley stadium, and london is in england** , where david cameron is the former leader .
GraphWriter	**the elizabeth tower is in london and the wembley stadium is in london , england** . london is in england and the former leader is david cameron .
GCP (proposed)	the wembley stadium and elizabeth tower are both located in london , england . the former leader of england is david cameron .

TABLE 4
Human evaluation results.

Model	Seen			Unseen			GKB		
	Correctness	Grammar	Fluency	Correctness	Grammar	Fluency	Correctness	Grammar	Fluency
S-LSTM	2.13	2.27	2.14	1.44	1.82	1.58	1.48	1.97	1.74
T-LSTM	2.32	2.42	2.45	1.52	1.62	1.79	2.13	2.31	2.21
S-TRANS	2.20	2.35	2.16	1.45	1.84	1.67	1.53	2.01	1.79
T-TRANS	2.39	2.45	2.52	1.54	1.69	1.83	2.20	2.37	2.31
TFF	1.82	1.74	1.63	1.31	1.58	1.55	1.72	1.87	2.06
GCN	2.54	2.50	2.31	1.65	1.87	1.91	2.20	2.36	2.15
GraphWriter	2.64	2.56	2.38	1.66	1.90	1.96	2.25	2.42	2.24
GCP (proposed)	**2.69**	**2.58**	**2.52**	**1.87**	**2.01**	**2.03**	**2.28**	**2.52**	**2.37**

4.5 Human Evaluation

To complement the automatic evaluation, we conduct human evaluations for all of the masked models. We recruited ten human annotators. Each of them has studied English for at least ten years and completed education in a fully English environment for at least two years. We provide a website that shows them the triples and the generated text. The annotators are given training on the scoring criteria. We also provide scoring examples. We randomly selected 200 sets of triples along with the output of each model. We only select the sets with more than two triples. We use three evaluation metrics [52]: *correctness*, *grammaticality*, and *fluency*. For each pair of a triple set and generated sentences, the annotators give a score between one to three for each metric.

Correctness measures the semantics of the output. Scores 3, 2, and 1 are given to the generated sentences that contain zero, one, and more than one errors in the relationships between entities, respectively. *Grammaticality* measures the grammatical and spelling errors of the output. Similar to the *correctness* metric, scores 3, 2, and 1 are given to generated sentences with zero, one, and more than one grammatical and spelling errors, respectively. The last metric, *fluency*, measures the fluency of the output (e.g., contain repetition or not). The annotators give a score based on the aggregation of the sentences and the existence of sentence repetition. The total time spent for these evaluations is around 300 hours. Table 4 shows the results. The inter-annotator agreement measured by Fleiss' kappa [53] is 0.56, which indicates moderate agreement. The results confirm the automatic evaluation in which GCP achieves the best scores.

TABLE 5
The results of different content-planning mechanism.

Model	BLEU ↑		METEOR ↑	
	Seen	Shuffled	Seen	Shuffled
Without entity masking				
TransE-based	52.79±.18	52.31±.15	36.59±.26	35.00±.13
Random	51.17±.13	51.21±.28	34.85±.25	34.93±.30
Manual (GTR-LSTM)	**54.27**±.21	51.53±.13	**37.35**±.15	35.37±.20
Proposed (GCP)	53.99±.16	**53.73**±.19	37.25±.12	**37.05**±.18
- w/o WE graph	50.15±.29	50.10±.17	33.27±.18	33.05±.22
- w/o EO triples	50.35±.26	50.21±.15	34.27±.16	34.05±.25
- w/o COOC triples	53.67±.18	53.72±.14	36.67±.17	36.72±.17
With entity masking				
TransE-based	56.38±.13	56.19±.19	40.19±.15	39.84±.14
Random	55.24±.28	55.24±.20	38.66±.12	37.96±.27
Manual (GTR-LSTM)	58.32±.15	56.34±.22	40.81±.17	38.99±.13
Proposed (GCP)	**58.53**±.11	**58.32**±.15	**40.93**±.21	**40.60**±.15
- w/o WE graph	55.04±.21	54.83±.24	37.88±.19	37.23±.17
- w/o EO triples	56.54±.25	55.14±.26	38.54±.19	37.89±.18
- w/o COOC triples	57.97±.16	57.88±.22	40.85±.18	40.62±.12

4.6 Effect of Content-planning

The triples in the Seen test dataset are manually ordered, such that the order of the entities in the input triples follows the order of entity mentions in the target sentence. In real applications, the triples may be automatically gathered (e.g., for question answering), where the entities and triples may not follow their order of appearances in the target sentence. To evaluate the robustness of our model against such scenarios, we randomly shuffle the triples of the Seen dataset (de-

noted by **Shuffled** test dataset). We compare using different content-planning mechanisms for GCP. They use different topological sort tie-breaking procedures as follows.

- **TransE-based** content-planning, which uses the same tie-breaking procedure as the proposed full GCP model, but it learns the entity embeddings using TransE [19] instead of TransR in GCP.
- **Random** content-planning, which randomly selects an entity on its tie-breaking procedure.
- **Manual (GTR-LSTM)** content-planning, which breaks ties based on the order of entity mentions in the target sentence from the input triples. When such an order is unknown, it falls back to a random ordering. This variant is essentially the **GTR-LSTM** model, which is a graph encoder in our previous conference paper [18].
- **Proposed** content-planning, which uses all proposed features as detailed in Section 3.4

Our proposed embedding model is designed to capture the relationships between both entities and words while preserving the order of entity mentions in a sentence. The latter feature helps the tie-breaking procedure of GCP (Section 3.4). The experimental results in Table 5 show the effectiveness of the proposed tie-breaking procedure. Our proposed model outperforms the Random and TransE-based content-planners consistently. This demonstrates the effectiveness of our model in learning the order of entity mentions in the target sentences, as neither Random nor TransE-based content-planners considers such orders. To confirm these results, we further compare the performance of TransE and our model in predicting the entity order in a sentence. We randomly sample 500 pairs of input triples and the corresponding sentence output. For each input triples, we run the entity-aware topological traversal with both tie-breaking procedures. Then, we manually check whether the entity order generated by the traversal is the same as the entity mentioning order in the sentence output. TransE only yields 75.8%, while that for our model is 87.3%.

Comparing with the Manual content-planner, our model also yields better performance in general. On the Seen test dataset, both models achieve comparable performance. These results are expected since the triples in the Seen test dataset are ordered according to the entity mentions in the target sentences. However, on the Shuffled and GKB test dataset, the performance of the Manual content-planner drops substantially, while our proposed model is robust to such data and generates sentences of higher quality.

We further perform ablation tests to show the effectiveness of our entity-order aware embeddings. First, we remove the joint embedding learning, i.e., remove word entity graph and learn the embeddings separately (denotes as **w/o WE graph**). Next, we test the joint embeddings without any triple enrichment, i.e., no ordering or co-occurrence is considered (denoted as **w/o EO triples**). Finally, we evaluate the joint embeddings without co-occurrence triple enrichment, i.e., considering entity order but not co-occurrence (denoted as **w/o COOC triples**). The results in Table 5 show that the first two variants suffer in the performance substantially. Meanwhile, the exclusion of co-occurrence triples has a relatively smaller impact. These results show that our proposed entity-order aware embedding model effectively captures the proper entity order commonly exhibited in natural sentences.

5 Conclusions and Future Work

We proposed a graph encoder named GCP for sentence generation from knowledge base triples. The proposed model maintains the structure of input triples as a graph (i.e., a KG) to optimize the amount of information preserved in the input. The model can handle cycles to capture the global information of a KG and handle multiple relationships between entities of a KG. We propose an entity-order aware topological traversal to integrate the content-planning in the encoding process. Hence, our model not only works as an encoder but also serves as a content-planner. Our entity-order aware traversal algorithm helps to encode a KG in a more natural order by taking supervision signals on the processing order from entity-order aware embeddings trained over a word-entity graph.

The experimental results show that our GCP model offers better performance than all the competitors. On the Seen WebNLG dataset, our model outperforms the best existing model, the GraphWriter model, by up to 3.6%, 4.1%, and 3.8% in terms of BLEU, METEOR, and TER scores, respectively. On the GKB dataset, our model outperforms the GraphWriter model by up to 7.1%, 3.1%, and 2.2% in these three metrics, respectively.

Our learned entity-order aware embeddings may capture a kind of linguistic bias: the common entity order in natural sentences (learned from Wikipedia), even though there could be other correct entity orders. The common entity order should be sufficient for general-purpose scenarios. The consistency in the entity ordering makes reading the generated sentences easier. For future work, it would also be interesting to investigate poetry generation or story generation that may require diversity in the generated output.

Acknowledgments

Bayu Distiawan Trisedya is supported by the Indonesian Endowment Fund for Education (LPDP). This work is supported by Australian Research Council (ARC) Discovery Project DP180102050.

References

[1] Q. Wang, Z. Mao, B. Wang, and L. Guo, "Knowledge graph embedding: A survey of approaches and applications," *TKDE*, vol. 29, no. 12, 2017.

[2] R. Zhang, B. D. Trisedy, M. Li, Y. Jiang, and J. Qi, "A comprehensive survey on knowledge graph entity alignment via representation learning," *CoRR abs/2103.15059*, 2021.

[3] E. Reiter and R. Dale, *Building natural language generation systems*. Cambridge University Press, 2000.

[4] R. Lebret, D. Grangier, and M. Auli, "Neural text generation from structured data with application to the biography domain," in *EMNLP*, 2016.

[5] Q. Wu, C. Shen, P. Wang, A. Dick, and A. v. d. Hengel, "Image captioning and visual question answering based on attributes and external knowledge," *TPAMI*, vol. 40, no. 6, 2018.

[6] P. Wang, Q. Wu, C. Shen, A. Dick, and A. van den Hengel, "Fvqa: Fact-based visual question answering," *TPAMI*, vol. 40, no. 10, 2018.

[7] K. Cho, B. van Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using rnn encoder–decoder for statistical machine translation," in *EMNLP*, 2014.

[8] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," in *ICLR*, 2015.

[9] P. Vougiouklis, H. Elsahar, L.-A. Kaffee, C. Gravier, F. Laforest, J. Hare, and E. Simperl, "Neural wikipedian: Generating textual summaries from knowledge base triples," *Journal of Web Semantics*, 2018.

[10] D. Marcheggiani and L. Perez-Beltrachini, "Deep graph convolutional encoders for structured data to text generation," in *INLG*, 2018.

[11] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *ICLR*, 2017.

[12] R. Koncel-Kedziorski, D. Bekal, Y. Luan, M. Lapata, and H. Hajishirzi, "Text generation from knowledge graphs with graph transformers," in *NAACL*, 2019.

[13] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, "Graph attention networks," in *ICLR*, 2018.

[14] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," in *NIPS*, 2017.

[15] S. Wiseman, S. M. Shieber, and A. M. Rush, "Challenges in data-to-document generation," in *EMNLP*, 2017.

[16] R. Puduppully, L. Dong, and M. Lapata, "Data-to-text generation with content selection and planning," in *AAAI*, 2019.

[17] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, 1997.

[18] B. D. Trisedya, J. Qi, R. Zhang, and W. Wang, "Gtr-lstm: A triple encoder for sentence generation from RDF data," in *ACL*, 2018.

[19] A. Bordes, N. Usunier, A. Garcia-Duran, J. Weston, and O. Yakhnenko, "Translating embeddings for modeling multi-relational data," in *NIPS*, 2013.

[20] B. D. Trisedya, J. Qi, and R. Zhang, "Entity alignment between knowledge graphs using attribute embeddings," in *AAAI*, 2019.

[21] B. D. Trisedya, G. Weikum, J. Qi, and R. Zhang, "Neural relation extraction for knowledge base enrichment," in *ACL*, 2019.

[22] P. A. Duboue and K. R. Mckeown, "Empirically estimating order constraints for content planning in generation," in *ACL*, 2001.

[23] W. Lu and H. T. Ng, "A probabilistic forest-to-string model for language generation from typed lambda calculus expressions," in *EMNLP*, 2011.

[24] W. Lu, H. T. Ng, and W. S. Lee, "Natural language generation with tree conditional random fields," in *EMNLP*, 2009.

[25] K. Bontcheva and Y. Wilks, "Automatic report generation from ontologies: The miakt approach," in *NLDB*, 2004.

[26] P. Cimiano, J. Lüker, D. Nagel, and C. Unger, "Exploiting ontology lexica for generating natural language texts from RDF data," in *ENLG*, 2013.

[27] D. Duma and E. Klein, "Generating natural language from linked-data: Unsupervised template extraction," in *IWCS*, 2013.

[28] T. Liu, K. Wang, L. Sha, Z. Sui, and B. Chang, "Table-to-text generation by structure-aware seq2seq learning," in *AAAI*, 2018.

[29] L. Sha, L. Mou, T. Liu, P. Poupart, S. Li, B. Chang, and Z. Sui, "Order-planning neural text generation from structured data," in *AAAI*, 2018.

[30] O. Vinyals, M. Fortunato, and N. Jaitly, "Pointer networks," in *NIPS*, 2015.

[31] B. D. Trisedya, J. Qi, and R. Zhang, "Sentence generation for entity description with content-plan attention." in *AAAI*, 2020.

[32] Z. Wang, J. Zhang, J. Feng, and Z. Chen, "Knowledge graph and text jointly embedding," in *EMNLP*, 2014.

[33] I. Yamada, H. Shindo, H. Takeda, and Y. Takefuji, "Joint learning of the embedding of words and entities for named entity disambiguation." in *CoNLL*, 2016.

[34] H. Zhong, J. Zhang, Z. Wang, H. Wan, and Z. Chen, "Aligning knowledge and text embeddings by entity descriptions," in *EMNLP*, 2015.

[35] Y. Cao, L. Huang, H. Ji, X. Chen, and J. Li, "Bridge text and knowledge by learning multi-prototype entity mention embedding," in *ACL*, 2017.

[36] J. Lehmann, R. Isele, M. Jakob, A. Jentzsch, D. Kontokostas, P. N. Mendes, S. Hellmann, M. Morsey, P. van Kleef, S. Auer, and C. Bizer, "Dbpedia - a large-scale, multilingual knowledge base extracted from wikipedia," *Semantic Web*, 2015.

[37] C. Fellbaum, *WordNet: An Electronic Lexical Database*. MIT Press, 1998.

[38] A. S. Schwartz and M. A. Hearst, "A simple algorithm for identifying abbreviation definitions in biomedical text," in *Pacific Symposium on Biocomputing*, 2003.

[39] Z. Wang, J. Zhang, J. Feng, and Z. Chen, "Knowledge graph embedding by translating on hyperplanes," in *AAAI*, 2014.

[40] Y. Lin, Z. Liu, M. Sun, Y. Liu, and X. Zhu, "Learning entity and relation embeddings for knowledge graph completion," in *AAAI*, 2015.

[41] G. Ji, K. Liu, S. He, and J. Zhao, "Knowledge graph completion with adaptive sparse transfer matrix," in *AAAI*, 2016.

[42] K. S. Tai, R. Socher, and C. D. Manning, "Improved semantic representations from tree-structured long short-term memory networks," in *IJCNLP*, 2015.

[43] X. Liang, X. Shen, J. Feng, L. Lin, and S. Yan, "Semantic object parsing with graph lstm," in *ECCV*, 2016.

[44] T. Luong, H. Pham, and C. D. Manning, "Effective approaches to attention-based neural machine translation," in *EMNLP*, 2015.

[45] R. Nallapati, B. Zhou, C. Gulcehre, B. Xiang *et al.*, "Abstractive text summarization using sequence-to-sequence rnns and beyond," in *SIGNLL*, 2016.

[46] C. Gardent, A. Shimorina, S. Narayan, and L. Perez-Beltrachini, "Creating training corpora for nlg micro-planners," in *ACL*, 2017.

[47] K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu, "Bleu: a method for automatic evaluation of machine translation," in *ACL*, 2002.

[48] M. J. Denkowski and A. Lavie, "Meteor 1.3: Automatic metric for reliable optimization and evaluation of machine translation systems," in *WMT*, 2011.

[49] M. Snover, B. Dorr, R. Schwartz, L. Micciulla, and J. Makhoul, "A study of translation edit rate with targeted human annotation," in *AMTA*, 2006.

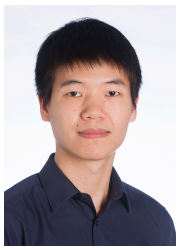
[50] J. H. Clark, C. Dyer, A. Lavie, and N. A. Smith, "Better hypothesis testing for statistical machine translation: Controlling for optimizer instability," in *ACL*, 2011.

[51] D. P. Kingma and J. L. Ba, "Adam: A method for stochastic optimization," in *ICLR*, 2015.

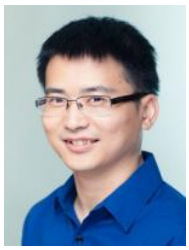
[52] C. Gardent, A. Shimorina, S. Narayan, and L. Perez-Beltrachini, "The webnlg challenge: Generating text from RDF data," in *INLG*, 2017.

[53] J. L. Fleiss, "Measuring nominal scale agreement among many raters." *Psychological Bulletin*, 1971.

Bayu Distiawan Trisedya is a Lecturer in the Faculty of Computer Science Universitas Indonesia, who is currently a Postdoctoral Research Fellow in the School of Computing and Information Systems at The University of Melbourne. He received his Bachelor's and Master's degrees from Universitas Indonesia in 2009 and 2011, respectively. He received his Ph.D. degree from The University of Melbourne in 2021.



Jianzhong Qi is a Senior Lecturer in the School of Computing and Information Systems at The University of Melbourne. He received his Ph.D. degree from The University of Melbourne in 2014. His research interests include machine learning and data management and analytics, with a focus on spatial, temporal, and textual data.



Wei Wang is currently a Professor in the Information Hub, The Hong Kong University of Science and Technology (Guangzhou). Prior to that, He was a Professor in the School of Computer Science and Engineering, University of New South Wales, Australia. He received the Ph.D. degree from the Hong Kong University of Science and Technology in 2004.

Rui Zhang is a Visiting Professor at Tsinghua University. His research interests include big data, data mining, and machine learning. Professor Zhang has won several awards, including Future Fellowship by the Australian Research Council in 2012, Chris Wallace Award for Outstanding Research by the Computing Research and Education Association of Australasia in 2015, and Google Faculty Research Award in 2017.