Extend, don't rebuild: Phrasing conditional graph modification as autoregressive sequence labelling

Leon Weber[Δ,□] and **Samuele Garda**[Δ] and **Jannes Münchmeyer**[◇,Δ] and **Ulf Leser**[Δ]
[Δ] Humboldt-Universität zu Berlin, [□] Max Delbrück Center for Molecular Medicine,
[◇] GFZ German Research Center for Geoscience Potsdam
weberple@hu-berlin, gardasam@hu-berlin.de,
munchmej@gfz-potsdam.de, leser@informatik.hu-berlin.de

Abstract

Deriving and modifying graphs from natural language text has become a versatile basis technology for information extraction with applications in many subfields, such as semantic parsing or knowledge graph construction. A recent work used this technique for modifying scene graphs (He et al., 2020), by first encoding the original graph and then generating the modified one based on this encoding. In this work, we show that we can considerably increase performance on this problem by phrasing it as graph extension instead of graph generation. We propose the first model for the resulting graph extension problem based on autoregressive sequence labelling. On three scene graph modification data sets, this formulation leads to improvements in accuracy over the state-of-the-art between 13 and 26 percentage points. Furthermore, we introduce a novel data set from the biomedical domain which has much larger linguistic variability and more complex graphs than the scene graph modification data sets. For this data set, the state-of-the art fails to generalize, while our model can produce meaningful predictions.

1 Introduction

Generating or modifying graphs based on natural language texts is a versatile technique that has applications in different subfields of Natural Language Processing (NLP) such as dependency parsing (Manning and Schütze, 2001) or semantic parsing (Oepen et al., 2019, 2020). However, while these tasks can all be viewed as instantiations of conditional graph generation, they have been traditionally addressed as distinct tasks with different data sets, models and evaluation settings. Contrary to that, we are interested in studying the general task of generating or modifying graphs based on textual input. Specifically, we focus on the recently introduced task of *conditional graph modification*, in which a model is given a graph which it should modify according to natural language instructions (He et al., 2020). Their proposed method first embeds both the graph and the instructions with a joint encoder into an embedding **h** and then rebuilds the graph using a separate generative model for graphs (You et al., 2018b) conditioned on **h**. While this approach achieves state-of-the-art results for the Scene Graph Modification (SGM) data sets, we identified two shortcomings of this approach: (i) The model has to newly generate also the parts of the input graph that actually should be left unmodified and (ii) the model uses a separate graph encoder in the generative decoder model, which does not share knowledge with the encoder.

We propose an alternative formulation of this problem in which we model the modification as a graph extension instead of graph generation. To this end, we introduce the two special node labels `ADD` and `DEL` which allow us to model node insertions, deletions and edge modifications in the graph extension setting. We develop a model for this novel graph extension problem that autoregressively solves a sequence labelling task for each node that is added to the graph. This formulation addresses both shortcomings of the model of He et al. (2020). First, it precisely extends the graph without the need for rebuilding the unmodified parts. Second, it models the graph as text which allows to encode the input text, the original graph and the extension with the same encoder and enables the straightforward integration of pretrained language models such as BERT (Devlin et al., 2019). Our proposed model outperforms the state-of-the-art for the three data sets published by He et al. (2020) by a large margin, with improvements between 13 and 26 percentage points (pp). To test the limits of our approach, we furthermore present a new, more challenging graph modification task in which biomedical event graphs have to be modified based on scientific texts. To this end, we transform data from an existing biomed-

Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing, pages 1213–1224
November 7–11, 2021.

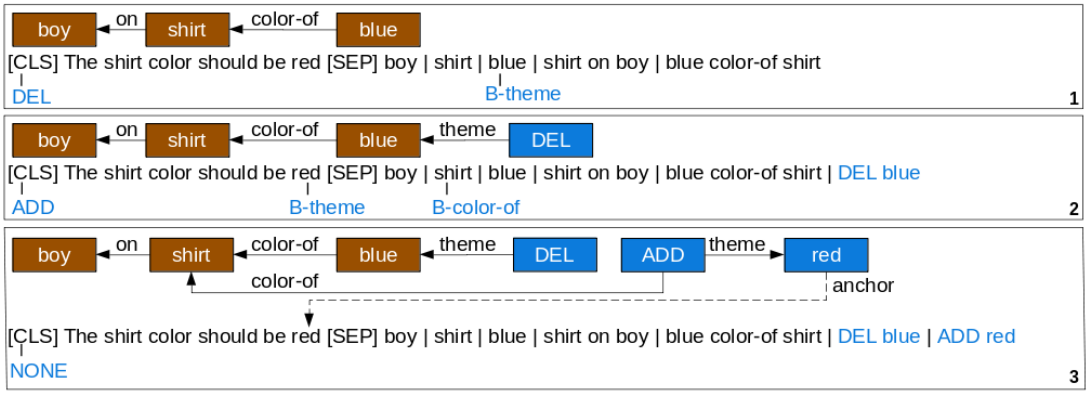

Figure 1: Rephrasing conditional graph modification as autoregressive sequence labelling. The substitution of the 'blue' node is replaced by the extension with the two nodes 'DEL' and 'ADD'. This graph extension problem is phrased as three autoregressive steps of sequence labelling. The original nodes have brown background while the extension nodes are drawn in blue.

ical event extraction task (Ohta et al., 2013) to a graph modification data set. Compared to the SGM data sets, the resulting data set displays much larger linguistic variation in the instruction texts and more complex graph structures. Our experiments show that the state-of-the-art fails to generalize on this data set, while our model is able to produce meaningful predictions. We analyze our model via a detailed ablation study and analyze the errors with respect to the input complexity, which allows us to precisely explain the improvements over state-of-the-art and suggest routes for even better future models.

To encourage further research on the challenging task of graph modification, we implement the models and data sets in a modular fashion and make the code available under an open-source license[1].

2 Related Work

Generating graphs from natural language texts is a central problem in many subfields of NLP.

Many classical problems in NLP such as dependency parsing (Manning and Schütze, 2001) or relation classification (Vu et al., 2016) are text-to-graph problems, but with highly restricted graph structures (e.g. nodes can be only words or named entities respectively). The methods developed for these tasks are typically tailored to these structures and cannot be used for other types of graphs. In contrast, our proposed method can handle arbitrary directed acyclic graphs (DAGs).

There is also significant interest in developing methods that jointly embed graphs and text to exploit graph-based information in NLP, especially for Question Answering based on Knowledge Bases (Lin et al., 2019; Yasunaga et al., 2021). However, these usually treat the graph as a static source of information and cannot be used for generating or extending graphs.

A task closer to our work in this regard is Cross-Framework Meaning Representation Parsing (Oepen et al., 2019, 2020) (MRP). In this task, systems are required to parse text into a general graph-based format in which nodes are not necessarily anchored in the text. The major difference between MRP and our graph modification setting is that in MRP the models always generate the full graph from scratch, while our method modifies an already provided graph. There is strong interest in graph generation also outside the NLP community, e.g. for modelling arbitrary distributions of graphs (You et al., 2018b; Liao et al., 2019) or for generating novel protein structures (Jin et al., 2018; You et al., 2018a). However, these methods neither do graph generation conditioned on textual input nor support the modification of partial graphs.

To the best of our knowledge, the only model that was explicitly developed for modifying arbitrary graphs based on natural language instructions is the model by He et al. (2020). It uses a transformer (Vaswani et al., 2017) to jointly embed text and graph, modelling the graph structure by restricting attention only to neighbouring nodes and by adding edge label embeddings onto the node embeddings. Based on this joint embedding, the modified graph is generated by a separate decoder based on the GraphRNN architecture (You et al.,

[1] https://github.com/leonweber/extend

2018b). While this architecture allows to model graph modification as graph generation, it also requires the model to generate the unmodified parts of the graph again which leaves more room for errors, whereas we only extend the graph leaving the unmodified parts untouched. Furthermore, this formulation uses two separate graph encoders; the transformer for encoding the original graph and the GraphRNN for encoding the partially generated graph. In contrast, our proposed method uses the same encoder for encoding the given graph and its extensions, allowing for more parameter sharing and thus for potentially better graph representations. Our simpler architecture makes the integration of BERT-style pretrained language models (Devlin et al., 2019; Gu et al., 2020) straightforward, however this only partly explains the observed gains over He et al. (2020) (see Section 5.3).

3 Methods

The task of graph modification is to modify a graph $\mathcal{G}$ into a graph $\mathcal{G}'$ according to natural language instructions t. To this end, let $\mathcal{G} = (\mathcal{N}, \mathcal{E})$ be a DAG consisting of nodes $\mathcal{N}$ and edges $\mathcal{E} \subseteq \mathcal{N} \times \mathcal{N}$, along with node labels $l_n \in \mathcal{L}_n \cup \{\texttt{NONE}\}$ and edge labels $l_e \in \mathcal{L}_e \cup \{\texttt{NONE}\}$. Let $\mathcal{G}'$ be defined analogously. We develop a model to estimate $p(\mathcal{G}' \mid t, \mathcal{G})$. Here, we first present our approach for the problem of graph extension, i.e., the case $\mathcal{N} \subseteq \mathcal{N}'$ and $\mathcal{E} \subseteq \mathcal{E}'$.

Then, we show how general graph modification can be reduced to this case. To this end, we identify three ways in which a graph can be modified: a node can be added, a node can be removed or the edges of an existing node can be modified. We show how we can model all three cases by introducing the two special node labels `ADD` and `DEL` which can be used to model node insertions and deletions in a graph extension setting. Both, `ADD` and `DEL` identify the argument that should be added or deleted via a special `theme` edge.

3.1 Explanation by Example

A visual explanation of our method can be found in Figure 1 and we will use it as a running example in the text. In this graph, we have the three nodes `boy`, `shirt` and `blue` and the edges (`shirt, on, boy`) and (`blue, color-of, shirt`). We want to change the color of the shirt from blue to red. For this, we extend $\mathcal{G}$ by a `DEL` node with its edge (`DEL, theme, blue`) and a `red` node with its edge (`red, color-of, shirt`). In our autoregressive sequence labelling framework, this extension of the original graph by two nodes is modelled as three successive calls to a sequence labelling model that receives as input the input text and a linearized form of the current graph. In the first call to the sequence labelling model, the input consists of the text and a linearization of the original graph. As output, the model produces the first extension node with label `DEL` and one edge to the node with the label `blue` (`DEL, theme, blue`). This is achieved by labelling the *CLS* token as `DEL` and the linearized representation of the `blue` node with `theme`. In the next step, the model receives as input the text and a linearized form of the now partially extended graph and predicts the next extension by labelling *CLS* with `ADD` to predict the node label. Additionally, the model predicts the two edges (`ADD, theme, red`) and (`red, color-of, shirt`). The predicted edge (`ADD, theme, red`) reflects the addition of a new node with the label `red` and is modelled by labeling the word *red* in the text with `self`. Note that this also produces an anchoring of the `red` node to the labeled span in the text. The edge (`red, color-of, shirt`) is produced by labeling the linearized form of the `shirt` node with `color-of`. After receiving the text and the further extended graph as input, the model signals the end of the extension process by labeling *CLS* with `NONE`.

3.2 Graph Extension as Autoregressive Sequence Labelling

We formulate the graph extension problem autoregressively. That is, we extend the provided graph one node at a time, together with the corresponding edges starting at the node. Let $\mathcal{N}^+ = \mathcal{N}' \setminus \mathcal{N}$ be the extension nodes in our graph, and $\pi = (n_1^+, n_2^+, ..., n_N^+)$ an ordering thereon. Note that the size $N = |\mathcal{N}^+|$ of the extended graph is known during training, while at test time we abort the generative process after a fixed number of steps which we treat as a hyperparameter or when the model predicts a node with label `NONE`. We write $\mathcal{G}'_i = \mathcal{G}'[\mathcal{N} \cup \{n^+_{j \leq i}\}]$ for the subgraph induced by the union of all nodes $\mathcal{N}$ from the original graph with the additional nodes up to n_i^+ in π. Let

$$p(\mathcal{G}', \pi \mid t, \mathcal{G}) = \prod_{i=1}^{N} p(n_i^+, \{e_{ij}\}_j \mid t, \mathcal{G}'_{i-1}), \quad (1)$$

be the joint probability of $\mathcal{G}'$ and the ordering π, where $\{e_{ij}\}_j$ are all edges from n_i^+ to all available nodes n_j. As multiple orderings can lead to the same graph, the total probability of a graph $\mathcal{G}'$ is

$$p(\mathcal{G}' \mid t, \mathcal{G}) = \sum_{\pi} p(\mathcal{G}', \pi \mid t, \mathcal{G}) \quad (2)$$

where the sum is over all possible orderings.

In practice, marginalising over all possible orderings is infeasible. Therefore, we impose two conditions on the ordering to reduce their number, often even making it unique. First, π must be a topological ordering, which exists as $\mathcal{G}'$ is a DAG. Second, among the topologically sorted orderings, we impose an additional order of node labels. In our running example, there are two possible topological orderings of the extended graph, because both extensions nodes could be added first. However, we impose that `DEL` nodes always come before `ADD` nodes, making π unique in this case.

We estimate $p(n_i^+, \{e_{ij}\}_j \mid t, \mathcal{G}'_{i-1})$ by formulating it as a sequence labelling task over a combination of the provided text t and a textual representation of $\mathcal{G}'_{i-1}$. We solve the resulting sequence labelling task with BERT. For this, we first linearize $\mathcal{G}'_i$ into a textual representation $t_{\mathcal{G}'_i}$. We treat the exact form of the linearization as a hyperparameter, with the only constraint that every node $n \in \mathcal{G}'_i$ is represented by a unique span denoted *span*(n). A possible linearization for the original graph in our running example would be `boy | shirt | blue | shirt on boy | blue color-of-shirt`. We use the linearization to jointly predict the label of the added node n_i^+ and its edges $\{e_{ij}\}_j$. To generate the prediction, we concatenate the instruction text t and the linearized graph $t_{\mathcal{G}'_i}$ in the form "[CLS] t [SEP] $t_{\mathcal{G}'_i}$". The model predicts the label of n_i^+ using the embedding of BERT's *[CLS]* token $h_{[CLS]}$. Then, the model predicts the labels of the outgoing edges of n_i^+ to all possible target nodes j. This is achieved by marking j's span in $t_{\mathcal{G}'_i}$ with the corresponding edge label (including `NONE`) in an IOB-tagging scheme.

That is, for producing a sequence labelling that represents the addition of the `DEL` node with its edge (`DEL, theme, blue`), we would mark the *[CLS]* token as `DEL` and the token `blue` as `B-theme`, with all other tokens being labelled as `O`.

We then estimate the joint probability of n_i^+'s label and the labels of the edges e_{ij} from n_i^+ to all possible target nodes j, by conditioning the edge probabilities on the node label. To this end, we first predict the node label from the embedding of the *[CLS]* token :

$$p(n_i^+ \mid t, \mathcal{G}'_{i-1}) = \texttt{softmax}(\mathbf{W}_{\mathcal{N}} \cdot \mathbf{h}_{[\mathbf{CLS}]}) \quad (3)$$

We then predict the single edge probabilities conditioned on the node label:

$$p(e_{ij} | n_i^+, t, \mathcal{G}'_{i-1}) = \prod_{k \in \textit{span}(j)} \texttt{softmax}(\mathbf{W}_{\mathcal{E}}^{(n_i^+)} \cdot \mathbf{h}_{\mathbf{k}}), \quad (4)$$

where $\mathbf{h}_k$ are the token embeddings of j's span with k ranging over each token. For modelling the joint probability, we assume independence between the edges:

$$p(n_i^+, \{e_{ij}\}_j \mid t, \mathcal{G}'_{i-1}) = p(n_i^+ \mid t, \mathcal{G}'_{i-1}) \cdot \prod_j p(e_{ij} | n_i^+, t, \mathcal{G}'_{i-1}), \quad (5)$$

where $\mathbf{W}_{\mathcal{N}}$ is the node-classification layer and $\mathbf{W}_{\mathcal{E}}^{(n_i^+)}$ the edge classification layers for each node label.

For training, we use the negative-log likelihood $-\log p(n_i^+, \{e_{ij}\}_j \mid t, \mathcal{G}'_{i-1})$ as loss, together with teacher forcing (Williams and Zipser, 1989). For prediction, we employ greedy search, choosing $\arg\max p(n_i^+, \{e_{ij}\}_j \mid t, \mathcal{G}'_{i-1})$ at each step.

In some applications, such as semantic parsing, it can be necessary to anchor some nodes to the text, i.e. assign a specific span in t to a node (Oepen et al., 2020). For instance, this can be used to encode that a certain semantic concept represented by a node is expressed in a specific span in the text. Our proposed autoregressive sequence labelling framework provides natural support for such a node anchoring, by including edges to spans in t. This is modelled by labelling the anchor spans in t with the desired edge type. Refer to Figure 1 for an example in which the `ADD` node, added in the second pane, has an edge to the span *red* in t. This edge triggers the creation of one additional node with label `red` which encodes the anchoring information.

3.3 Modelling Graph Modification as Graph Extension

We now formulate the graph modification problem as described in He et al. (2020) as a graph extension

task. In contrast to the formulation as graph generation, this framework does not require the model to reproduce the unmodified parts of the graph. We produce an extended graph $\mathcal{G}'$ from the original graph $\mathcal{G}$, which contains information on the modifications to apply. From this graph, an application-independent postprocessing can trivially calculate the modified graph $\mathcal{G}_m$.

We distinguish three different ways in which $\mathcal{G}$ can be modified: (1) an existing node n is deleted, (2) a node n is added and (3) the edges of an existing node n are changed.

For case (1), we add a *DEL* node with a single `theme` edge to n. In the post processing, we remove all nodes (and connected edges) that have edges from *DEL* nodes.

For case (2), we introduce an *ADD* node that adds n and all its outgoing edges. To determine the label of the added node, we extend $\mathcal{G}$ by another node representing the label of n. Modelling the label of added nodes in this way instead of predicting it directly, allows us to optionally use anchor nodes to determine the labels of added nodes. This can drastically reduce the size of the output space if the number of node labels is very large. For instance, in our running example, we want to add a node with the label *red*. The proposed model can achieve this in two ways: The first option is depicted in Figure 1, where it predicts the `ADD` node and marks the span "red" in t as the special `theme` edge. This is then interpreted as a prediction of the `ADD` node and that of an anchor node with label `red`. The second option is to first generate a node with label `red` and in a later generation step the `ADD` node with a `theme` edge to the `red` node. Both lead to the same graph (third pane in Figure 1), with the exception of the `anchor` edge, which would only be present under the first option. In the post processing, we change the label of n from *ADD* to the label of the additional label node and then remove the label node.

For case (3), we model modified edges of n as a sequence of deletions and additions by deleting n and then adding it back with the modified edges using the operations described in cases (1) and (2).

4 Experimental Setup

We evaluate our model on three data sets for SGM and a novel Biomedical Event Graph Completion data set.

4.1 Scene Graph Modification

SGM is a task defined by He et al. (2020). The model is given a scene graph and modification instructions in natural language and has to produce a new version of the graph that was modified according to the instructions. He et al. (2020) published three data sets: *MSCoco*, *GCC* and *CrowdSourced*. The first two were created synthetically from publicly available data sets (Lin et al., 2014; Sharma et al., 2018), while the instructions of the third were generated via crowd sourcing. Data set statistics can be found in Table 1.

	MSCoco	GCC	Crowd	PC13
Size	196k/2k/2k	400k/7k/7k	30k/1k/1k	4k/0.5k/1.5k
Avg. $\lvert\mathcal{N}\rvert$	3±0.9	4±1.9	2.0±0.8	5.9 ±4.4
Avg. $\lvert\mathcal{N}_m\rvert$	3±1.4	4±2.0	2.0±0.8	6.8 ±4.6
Avg. $\lvert\mathcal{E}\rvert$	2±1.0	3±2.0	1.0±0.8	3.5 ±4.6
Avg. $\lvert\mathcal{E}_m\rvert$	2±1.4	3±2.0	1.0±0.8	4.8 ±5.0
OOV t (%)	-/3/4	-/2/2	-/8/8	-/35/48
OOV $\mathcal{N}$ (%)	-/4/4	-/3/3	-/11/10	-/64/67

Table 1: Data set statistics before transformation to graph extension. We report mean and standard deviation for the graph statistics. OOV t and OOV $\mathcal{N}$ denote the percentage of text tokens and node labels from the development/test set that do not appear in the training set.

We rephrase the graph modification task as a graph extension problem as described in Section 3.3. We found that in these data sets the labels of almost all extension nodes appear in the modification prompts verbatim. Accordingly, we introduce additional anchoring nodes by exact string matching of the node label with the textual instructions. While this means that our model now has to add twice as many nodes (one anchor and one *ADD* node per additional node) it allows us to reduce the output space for the node label from 14,873 / 26,827 / 5,747 to two (`ADD` and `DEL`) for the training sets of *MSCoco*, *GCC* and *CrowdSourced* respectively.

The edges in all three SGM data sets are undirected but for our proposed framework we require the graph to be directed. Thus, we transform the undirected graphs to DAGs by defining the directions of edges between extension nodes and original nodes $\{n^+, n\}$ with $n^+ \in \mathcal{N}^+, n \in \mathcal{N}$ to go from n^+ to n. The rest of the directions is assigned arbitrarily.

For SGM, we linearize the graphs by writing out all nodes as each comes with a unique natural language label such as ‘shirt’ or ‘blue’. Additionally, we write out all (directed) edges (u, v) in the form

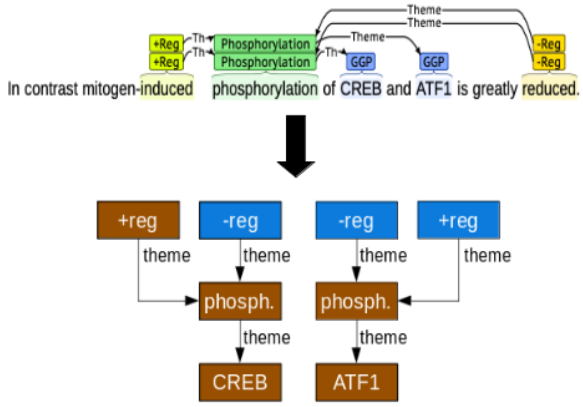

Figure 2: Creation of the PC13 data set. The event annotations are transformed into the TEG and some nodes are randomly chosen as extension nodes (here in blue).

`<u> <edge-label> <v>`. See Figure 1 for a detailed example.

4.2 Biomedical Event Graph Completion

Biomedical Event Extraction is an information extraction task in which events that model biomedical processes have to be extracted from text (Ohta et al., 2013). These events are defined by their trigger (a typed span in the text) and their arguments, which can be other events or named entities and have a type called role. Together, all events and named entities in a given text form a directed graph which we call Text Event Graph (TEG). The nodes of the TEG are comprised of all named entities and all events in the text with their provided labels. The edges in the TEG always originate from event nodes and can have other event nodes or entity nodes as targets. An example TEG can be found in Figure 2.

We transform a Biomedical Event Extraction data set to a graph modification data set by randomly deleting event nodes and asking the model to recover them. Specifically, we use the *BioNLP 2013 Pathway Curation* (PC13) dataset (Ohta et al., 2013), split it into sentences and then randomly delete between zero and three event nodes, with the constraint that no more than 75% of the events can be deleted. We treat event hedging (negation and speculation) as special event nodes with one edge to the modified events. Furthermore, we delete all triggers (which correspond to anchor nodes), so that we can also evaluate the method of (He et al., 2020) which does not have support for anchor nodes. Statistics of the resulting data set can be found in Table 1. Notably, the PC13 data set differs considerably from the three SGM data sets in important respects. First, the task presented by the data set is a pure graph extension task as it is only necessary to add nodes. Second, both original and modified graphs are much larger than the scene graphs both in terms of nodes and of edges. Third, the PC13 data set is much smaller in terms of examples. Fourth, the PC13 data set possesses a much larger linguistic variability which is reflected in a larger variability in node labels, because all named entities of the text appear as nodes in the graph. This leads to a very high number of node labels and words in the text which appear in the dev/test set but not in the training set (35-67% vs 8-11% in the *CrowdSourced* data set). Overall, we consider this data set as considerably more challenging than the SGM data, which is reflected in much lower performance (see Section 5) in our experiments. Importantly, to correctly modify the graph, the models frequently have to generate more than one node with multiple edges, making it harder to achieve a prediction that is correct on the graph level than for the SGM data sets, where the modifications are limited to a change of exactly one node.

For graph linearization, we first write out all entity nodes using their associated text attributes. We append a linearization of each event e that consists of its label together with all edges e, n in the form `<edge-label> ( <n> )`. If n is an event itself, we use its linearization, which is possible because the graph is free of cycles. An example linearization can be found in Appendix B.

4.3 Evaluation Metrics & Baselines

We follow He et al. (2020) and report the metrics graph accuracy, node F1 and edge F1. For graph accuracy, we define a predicted graph to be correct if it is isomorphic to the ground truth under the constraint that the labels of nodes and edges match. In the SGM data sets, the labels of the nodes are typically unique in a given graph and thus can be used to define precision and recall for nodes and edges in their standard formulation.

For PC13, there are usually multiple nodes with the same label, which makes it necessary to use an alternative definition for whether an extension node n^+ is present in some reference graph $\mathcal{G}_r$. We define that $n^+ \in \mathcal{G}_r$ if the subgraph induced by n^+ and its descendants is isomorphic to a subgraph in $\mathcal{G}_r$. Because each n^+ corresponds to an event and the event is fully specified by this descendant

subgraph, this formulation corresponds to the standard evaluation protocol in the BioNLP shared task series (Ohta et al., 2013) with the exception of anchor nodes which we disregard to allow for a fair comparison to He et al. (2020).

We compare our model on all data sets with the best configuration reported by He et al. (2020), which is their cross-attention model that jointly embeds text and graph with a transformer. As an additional baseline, we use the CopySource baseline which simply predicts the unmodified source graph.

4.4 Training Details

For the SGM data sets, we use the *bert-base-uncased* (Devlin et al., 2019) model of HuggingFace transformers (Wolf et al., 2019) as our pretrained transformer. We optimize our models with Adam (Kingma and Ba, 2015) using a batch size of 16 and a learning rate of 3e-5 for 100 epochs on CrowdSourced and for 20 epochs on the two other data sets.

For PC13, we use the HuggingFace transformers' version of *BiomedNLP-PubMedBERT-base-uncased-abstract-fulltext* (Gu et al., 2020), a version of BERT trained on biomedical texts, as the transformer and train for 100 epochs, using a batch size of 16 and a learning rate of 3e-5.

For all datasets, we abort the graph extension process after 10 generated nodes.

5 Results

5.1 Comparison with State of the Art for Scene Graph Modification

Results for the SGM data sets can be found in Table 2. On all four data sets, our proposed method outperforms the model of He et al. (2020) by 13 to 26 pp accuracy. The improvement is especially pronounced on the CrowdSourced data set. We attribute this stronger improvement to two characteristics of the data set that make it benefit from using pretrained language models. First, compared to the two other data sets, CrowdSourced is the only non-synthetic one which leads to a larger linguistic variability. Second, it is much smaller. For both characteristics, a pretrained language model such as BERT is an ideal solution, as it alleviates the need for large training data and was exposed to a lot of linguistic variation during pretraining. We verified this hypothesis by enriching the graph and text embeddings of the model of He et al. (2020) with fine-tuned BERT embeddings (see Appendix A for details) which led to an improvement of 10pp on CrowdSourced but led to diminished results on the two other SGM data sets.

To quantify the advantage that a graph extension formulation has over graph generation, we analyzed how many errors the model of He et al. (2020) made because it incorrectly reconstructed the subgraph that should be left unmodified. As the weights of the models reported in He et al. (2020) are not publicly available, we retrained a model using the authors' implementation and the reported choice of hyperparameters on the CrowdSourced data set. For the 1000 development examples, the resulting model produced 403 incorrect graphs. Almost 50% of these errors (181) were due to incorrect reconstructions of the original graph, whereas the proportion is only 16% for our model. This confirms our hypothesis that reformulating graph modification as graph extension instead of graph generation helps to avoid a large proportion of errors in reconstructing the parts of the graph that should be left unmodified.

5.2 Performance in BioNLP Event Graph Completion

Results for the PC13 data set can be found in Table 3. Our proposed method achieves a graph accuracy of 47.12% and improves upon the CopySource baseline by over 2pp. However, both in terms of Node F1 and Edge F1, CopySource performs better than our method, which indicates that when our model wrongly extends a graph it does this frequently by introducing more than one wrong node or edge. The model of He et al. (2020) fails to produce meaningful predictions, achieving only 1.09% accuracy. We attribute this to the high rates of tokens and node labels that appear in the test set but not in the training set (48% of the tokens and 67% of the node labels). Because this model attempts to reproduce the whole graph and because it treats each node label as a class, it has no appropriate mechanism for predicting modifications of graphs that have a large number of unknown node labels. Additionally, we expect the model to struggle with a large number of unknown tokens in the instruction, because it does not use pretrained embeddings. This hypothesis is supported by the fact that the model achieves 71.67% accuracy on the PC13 train set as opposed to the 1.09% on the test set. Note, that this failure to generalize cannot be explained

	Crowd Sourced			MSCoco			GCC		
	Node F1	Edge F1	Graph Acc	Node F1	Edge F1	Graph Acc	Node F1	Edge F1	Graph Acc
Copy Source	66.17	31.42	-	78.41	64.62	-	79.46*	66.32*	-
Text2Text	78.59	52.68	52.15	91.47	72.74	64.42	-	-	-
Mod. GraphRNN	80.68	57.17	56.75	80.64	55.76	50.72	-	-	-
Graph Transformer	81.47	59.43	58.23	91.21	75.68	71.38	-	-	-
DCGCN	79.05	54.23	52.67	89.08	72.47	68.89	-	-	-
He et al. (2020)	83.69	62.1	60.9	95.4	86.52	82.97	93.84	57.68	52.5
Ours	**97.62**	**88.26**	**87.6**	**99.52**	**98.4**	**96.15**	**98.62**	**91.64**	**75.01**
- BERT	95.44	82.13	82.7	99.36	98.2	95.45	98.52	91.55	73.66

Table 2: Comparison with state-of-the-art on the scene graph modification test sets. Baseline results are taken from He et al. (2020) including missing values for CopySource and results for Modified Graph RNN (You et al., 2018b), Graph Transformer (Cai and Lam, 2020) and DCGCN (Guo et al., 2019). Results marked with a '*' denote results obtained by us.

	Node F1	Edge F1	Graph Acc
CopySource	**93.59**	**84.67**	44.88
He et al. (2020)	45.08	3.41	1.09
Ours	91.38	80.54	**47.12**

Table 3: Results for the PC13 BioNLP-completion test set. Best results are in bold.

	CrowdSourced	PC13
CopySource	-	42.15
He et al. (2020)	61.2	2.07
Ours	87.60	**48.35**
- BERT	82.7	41.53
- anchors	83.7	-
- conditional loss	**88.1**	44.83
- edges in linearization	61.5	-
+ linearization in natural language	-	46.28

Table 4: Accuracy scores for ablations on the CrowdSourced and PC13 dev sets. Best results are in bold. '+' / '-' denote independent extensions / ablations of model components.

purely by the absence of a pretrained model component, because our proposed model still performs much better when the pretrained component is ablated (see Section 5.3).

5.3 Explaining the Performance Gains via Ablations

We performed an ablation study on the development sets of CrowdSourced and PC13 to identify the source of performance gains achieved by our model compared to He et al. (2020). Results can be found in Table 4. The ablation of the pretrained language model BERT, in which we used the same architecture as in our original model but initialized all parameters from scratch, led to a decrease in accuracy of 4.9pp on CrowdSourced and 6.8pp on PC13.

Note, that for PC13 the results without BERT are worse than the results of the CopySource baseline, which confirms our hypothesis that for this data set a pretrained language model is required to generalize well.

We also investigated how important the type of graph linearization is. To this end, we tested a variation of the linearization for each of the two data sets: For PC13, we changed the proposed linearization that contains all information about the event graph to text that is formulated closer to natural language, which has the downside that argument edges to other events may not be uniquely represented but might be easier to analyze by the language model. For instance, we would change `regulation cause ( stat1 ) theme ( pathway participant ( ifn - gamma ) )` to `regulation of pathway containing ifn-gamma by stat1`. This led to a decrease in accuracy of roughly 2pp, indicating that uniqueness and full information in the graph linearization might be more important than natural sounding language. For CrowdSource, the linearization is already natural sounding and unique. We evaluated a linearization without any edge representations, retaining only a list of the contained nodes, to test whether our proposed model makes use of information relating to the graph topology. This led to a pronounced drop in accuracy of roughly 26pp, which verifies that our proposed model makes use of the edge information and that a full representation of the graph is required for strong performance on this data set.

Furthermore, we checked whether the conditioning of the edges on the node label is beneficial, as it comes at the price of increasing the number of parameters in the output layer by a factor of $|\mathcal{L}_n|$. For this, we evaluate a variant of our model

in which we treat the prediction of node label and edge labels as independent. For PC13, this leads to a decrease of over 3pp in accuracy, which we expected, because the allowed edge labels differ strongly depending on the node label. On CrowdSourced, the ablation of this dependency actually improved accuracy by roughly 0.5pp. We hypothesize that this is because there are only two node labels which can be predicted in this data set (`ADD` and `DEL`) and thus there is no strong dependency relations between node and edge labels. This leads to redundant parameters in the output layer which have to be learned from the same amount of training data.

Finally, we suspected that a large factor of the improvements over the He et al. (2020) model is the introduction of anchors, which essentially transforms the generative task of predicting the node label to a discriminative sequence labeling task. To test this, we perform an ablation in which we remove all anchors and instead extend the graph with a node that has the appropriate label. As this drastically increases the number of node labels, conditioning the edge labels on the node labels leads to out-of-memory exceptions on a single Nvidia RTX 3090 with 24 GB of RAM. Thus, we use the unconditional probabilities mentioned in the ablation of the conditional probability. We found the the ablation of anchors indeed led to a notable drop of almost 4pp in accuracy but that other factors such as the pretrained language model and the graph linearization had a much larger effect.

5.4 Error analysis

We analyzed the performance for predicting missing nodes on the PC13 development set with respect to various characteristics of the input data. Note, that here, precision, recall and F1 are calculated with respect only to *extension* nodes, while node F1 is calculated with respect to all nodes.

First, we investigated the effect of error accumulation. To test this, we analyzed how the precision of a node behaves as a function of the step at which it was predicted (the index in π, see 3.2). Surprisingly, we did not find a clear trend with precision being roughly 51%, 33% and 50%, for the first, second and third step, respectively. However, there was a strong trend for decreasing recall with the number of missing event nodes in the graph with 54%, 34% and 17% for one, two and three missing nodes. This might be because of the small amount of training examples with multiple missing nodes (see Table 1) or due to error accumulation.

Additionally, we found a moderate negative correlation between the number of nodes in the input graph $\mathcal{G}$ and precision (*Pearson's* $r = -0.47$) and a stronger negative correlation between the number of nodes and recall (*Pearson's* $r = -0.71$). This indicates that one route to further improve our proposed model might be to strengthen its ability to reason about complex graphs.

We conjectured that `ADD` nodes would be much harder to predict than `DEL` nodes, because `DEL` nodes have exactly one edge with only one possible edge label (`theme`), whereas `ADD` nodes can have arbitrarily many edges. Indeed, we found that our proposed model achieved an F1 score of over 97% for `DEL` nodes, as opposed to 76% for `ADD` nodes on the development portion of the CrowdSourced data set, confirming our hypothesis.

6 Conclusion

We have developed a novel formulation of the conditional graph modification problem as conditional graph extension. This allows us to only generate the modified parts of the graph as opposed to rebuilding the full graph. Additionally, our model uses only one encoder for both graph and text allowing for maximum parameter sharing and can make use of pretrained language models such as BERT. On three SGM data sets and on a newly introduced biomedical event graph completion data the proposed model outperforms the state-of-the-art. Our error analysis highlights that performance degrades for larger input graphs. Thus, we plan to evaluate whether usage of more sophisticated Graph Neural Networks (Li et al., 2019) would improve results for these cases. We are also interested to apply our conditional graph modification framework to other tasks such as graph-based semantic parsing and knowledge graph completion, as this might yield a unified framework for many standard NLP tasks.

Acknowledgements

Leon Weber and Jannes Münchmeyer acknowledge the support of the Helmholtz Einstein International Berlin Research School in Data Science (HEIBRiDS). Samuele Garda is supported by the Deutsche Forschungsgemeinschaft as part of the research unit "Beyond the Exome".

References

Alan Akbik, Tanja Bergmann, Duncan Blythe, Kashif Rasul, Stefan Schweter, and Roland Vollgraf. 2019. FLAIR: An easy-to-use framework for state-of-the-art NLP. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics (Demonstrations)*, pages 54–59, Minneapolis, Minnesota. Association for Computational Linguistics.

Deng Cai and Wai Lam. 2020. Graph transformer for graph-to-sequence learning. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020*, pages 7464–7471. AAAI Press.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

Yu Gu, Robert Tinn, Hao Cheng, Michael Lucas, Naoto Usuyama, Xiaodong Liu, Tristan Naumann, Jianfeng Gao, and Hoifung Poon. 2020. Domain-specific language model pretraining for biomedical natural language processing. *CoRR*, abs/2007.15779.

Zhijiang Guo, Yan Zhang, Zhiyang Teng, and Wei Lu. 2019. Densely connected graph convolutional networks for graph-to-sequence learning. *Trans. Assoc. Comput. Linguistics*, 7:297–312.

Xuanli He, Quan Hung Tran, Gholamreza Haffari, Walter Chang, Zhe Lin, Trung Bui, Franck Dernoncourt, and Nhan Dam. 2020. Scene graph modification based on natural language commands. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: Findings, EMNLP 2020, Online Event, 16-20 November 2020*, pages 972–990. Association for Computational Linguistics.

Wengong Jin, Regina Barzilay, and Tommi S. Jaakkola. 2018. Junction tree variational autoencoder for molecular graph generation. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pages 2328–2337. PMLR.

Diederik P. Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*.

Guohao Li, Matthias Müller, Ali K. Thabet, and Bernard Ghanem. 2019. Deepgcns: Can gcns go as deep as cnns? In *2019 IEEE/CVF International Conference on Computer Vision, ICCV 2019, Seoul, Korea (South), October 27 - November 2, 2019*, pages 9266–9275. IEEE.

Renjie Liao, Yujia Li, Yang Song, Shenlong Wang, William L. Hamilton, David Duvenaud, Raquel Urtasun, and Richard S. Zemel. 2019. Efficient graph generation with graph recurrent attention networks. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 4257–4267.

Bill Yuchen Lin, Xinyue Chen, Jamin Chen, and Xiang Ren. 2019. Kagnet: Knowledge-aware graph networks for commonsense reasoning. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing, EMNLP-IJCNLP 2019, Hong Kong, China, November 3-7, 2019*, pages 2829–2839. Association for Computational Linguistics.

Tsung-Yi Lin, Michael Maire, Serge J. Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. 2014. Microsoft COCO: common objects in context. In *Computer Vision - ECCV 2014 - 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part V*, volume 8693 of *Lecture Notes in Computer Science*, pages 740–755. Springer.

Christopher D. Manning and Hinrich Schütze. 2001. *Foundations of statistical natural language processing*. MIT Press.

Stephan Oepen, Omri Abend, Lasha Abzianidze, Johan Bos, Jan Hajic, Daniel Hershcovich, Bin Li, Tim O'Gorman, Nianwen Xue, and Daniel Zeman. 2020. MRP 2020: The second shared task on cross-framework and cross-lingual meaning representation parsing. In *Proceedings of the CoNLL 2020 Shared Task: Cross-Framework Meaning Representation Parsing, CoNLL Shared Task 2020, Online, November 19-20, 2020*, pages 1–22. Association for Computational Linguistics.

Stephan Oepen, Omri Abend, Jan Hajic, Daniel Hershcovich, Marco Kuhlmann, Tim O'Gorman, Nianwen Xue, Jayeol Chun, Milan Straka, and Zdenka Uresová. 2019. MRP 2019: Cross-framework meaning representation parsing. In *Proceedings of the Shared Task on Cross-Framework Meaning Representation Parsing at the 2019 Conference on Natural Language Learning, CoNLL 2019, Hong Kong, Novemer 3, 2019*, pages 1–27. Association for Computational Linguistics.

Tomoko Ohta, Sampo Pyysalo, Rafal Rak, Andrew Rowley, Hong-Woo Chun, Sung-Jae Jung, Sung-Pil Choi, Sophia Ananiadou, and Jun'ichi Tsujii.

2013. Overview of the pathway curation (PC) task of BioNLP shared task 2013. In *Proceedings of the BioNLP Shared Task 2013 Workshop*, pages 67–75, Sofia, Bulgaria. Association for Computational Linguistics.

Piyush Sharma, Nan Ding, Sebastian Goodman, and Radu Soricut. 2018. Conceptual captions: A cleaned, hypernymed, image alt-text dataset for automatic image captioning. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics, ACL 2018, Melbourne, Australia, July 15-20, 2018, Volume 1: Long Papers*, pages 2556–2565. Association for Computational Linguistics.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 5998–6008. Curran Associates, Inc.

Ngoc Thang Vu, Heike Adel, Pankaj Gupta, and Hinrich Schütze. 2016. Combining recurrent and convolutional neural networks for relation classification. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 534–539, San Diego, California. Association for Computational Linguistics.

Ronald J. Williams and David Zipser. 1989. A learning algorithm for continually running fully recurrent neural networks. *Neural Comput.*, 1(2):270–280.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, R'emi Louf, Morgan Funtowicz, and Jamie Brew. 2019. Huggingface's transformers: State-of-the-art natural language processing. *ArXiv*, abs/1910.03771.

Michihiro Yasunaga, Hongyu Ren, Antoine Bosselut, Percy Liang, and Jure Leskovec. 2021. QA-GNN: reasoning with language models and knowledge graphs for question answering. *CoRR*, abs/2104.06378.

Jiaxuan You, Bowen Liu, Zhitao Ying, Vijay S. Pande, and Jure Leskovec. 2018a. Graph convolutional policy network for goal-directed molecular graph generation. In *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, pages 6412–6422.

Jiaxuan You, Rex Ying, Xiang Ren, William L. Hamilton, and Jure Leskovec. 2018b. Graphrnn: Generating realistic graphs with deep auto-regressive models. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pages 5694–5703. PMLR.

A Integrating BERT into He et al. (2020)

We evaluate a modified version of the model of He et al. (2020) in which we integrate a finetuned BERT component. To explain this modification, we use the notation of He et al. (2020) in which y ranges over the text tokens and x over the nodes of the unmodified graph. For this, we use Flair (Akbik et al., 2019) together with the *bert-base-uncased* model to calculate embeddings for tokens $\mathbf{h_y}$ and nodes $\mathbf{h_x}$. To represent nodes, we use the node label as input to BERT and if there are multiple subword tokens per token or node label, we use the embedding of the first. Then, we fuse the original token embeddings $\mathbf{m_y} \in \mathbb{R}^d$ and node embeddings $\mathbf{m_x} \in \mathbb{R}^d$ with two newly introduced single layer Multilayer Perceptrons:

$$\mathbf{m}'_y = \mathbf{W}_y^{(2)} \cdot (\mathit{ReLU}(\mathbf{W}_y^{(1)} \cdot [\mathbf{m_y}, \mathbf{h_y}])) \quad (6)$$

$$\mathbf{m}'_x = \mathbf{W}_x^{(2)} \cdot (\mathit{ReLU}(\mathbf{W}_x^{(1)} \cdot [\mathbf{m_x}, \mathbf{h_x}])), \quad (7)$$

where $\mathbf{W}_x^{(1)} \in \mathbb{R}^{d+768 \times d}$, $\mathbf{W}_x^{(2)} \in \mathbb{R}^{d \times d}$, $\mathbf{W}_y^{(1)} \in \mathbb{R}^{d+768 \times d}$, $\mathbf{W}_y^{(2)} \in \mathbb{R}^{d \times d}$ and BERT are the additional trainable parameters and $[\cdot]$ denotes concatenation. The resulting modified token embeddings $\mathbf{m}'_y$ and node embeddings $\mathbf{m}'_x$ are then used in place of the original ones leaving the rest of the implementation unchanged.

B Example of BioNLP TEG

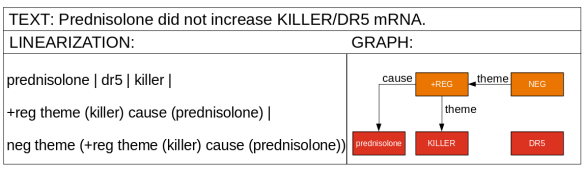

Figure 3: Illustration of a small BioNLP Text Event Graph, together with the associated text and the resulting graph linearization. Red squares are entities and orange boxes are events or event modifications.