

Everything Happens for a Reason: Discovering the Purpose of Actions in Procedural Text

Bhavana Dalvi Mishra*, Niket Tandon*, Antoine Bosselut,
Wen-tau Yih,† Peter Clark

Allen Institute for Artificial Intelligence, Seattle, WA
{bhavanad, nikett, antoineb, peterc}@allenai.org

Abstract

Our goal is to better comprehend procedural text, e.g., a paragraph about photosynthesis, by not only predicting what happens, but *why* some actions need to happen before others. Our approach builds on a prior process comprehension framework for predicting actions' effects, to also identify subsequent steps that those effects enable. We present our new model (XPAD) that biases effect predictions towards those that (1) explain more of the actions in the paragraph and (2) are more plausible with respect to background knowledge. We also extend an existing benchmark dataset for procedural text comprehension, ProPara, by adding the new task of explaining actions by predicting their dependencies. We find that XPAD significantly outperforms prior systems on this task, while maintaining the performance on the original task in ProPara. The dataset is available at <http://data.allenai.org/propara>

1 Introduction

Procedural text is common in natural language, for example in recipes, how-to guides, and science processes, but understanding it remains a major challenge. While there has been substantial prior work on extracting the sequence of actions (“scripts”) from such texts, the task of identifying *why* the sequence is the way it is has received less attention, and is the goal of this work. While “why” can mean many things, we treat it here as describing how one action produces effects that are required by another. For example in Figure 1, “CO₂ enters the leaf” is necessary because it results in “CO₂ is at the leaf”, a precondition for “CO₂ forms sugar”. (Note that

*Bhavana Dalvi Mishra and Niket Tandon contributed equally to this work.

†Scott is currently at Facebook AI Research (scott-yih@fb.com)

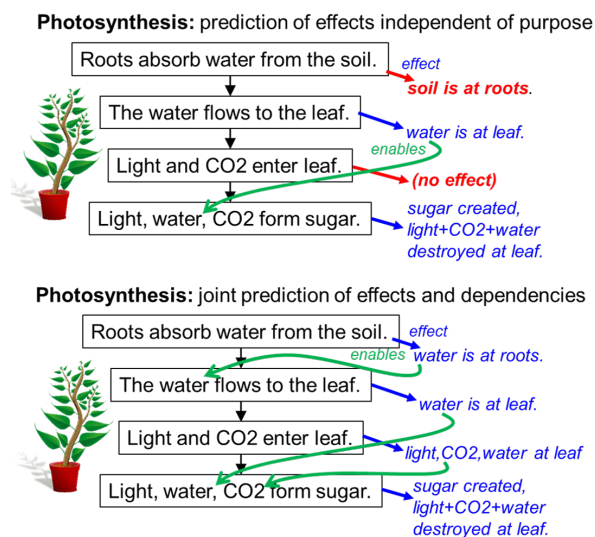


Figure 1: An action sequence (black) describes the order that actions happen, but not why. We add the new task of also predicting *why* the actions are needed, in the form of the actions' effects (blue) and subsequent actions that depend on those effects (green). While a system that predicts effects without considering dependencies can make errors (red, top part), we obtain better predictions by biasing the system to predict effects that *also* result in more dependencies (lower part).

an action does not directly depend on previous actions, but rather on the *state of the world* resulting from previous actions). If one could determine such rationales, new capabilities would become possible, including explanation, identifying alternative event orderings, and answering “*what if...*” questions. However, this task is challenging as it requires knowledge of the preconditions and effects of actions, typically unstated in the text itself.

Recent work in neural process modeling goes partway towards our goal by modeling the effects of actions, using annotated training data, allowing the states of entities to be tracked throughout a paragraph, e.g., EntNet (Henaff et al., 2017), NPN (Bosselut et al., 2018), and ProStruct (Tan-

don et al., 2018). However, these systems do not consider the purpose of those effects in the overall action sequence. As a result, they are unaware if they predict effects that have no apparent purpose in the process, possibly indicating a prediction error (e.g., the erroneous predictions in red in Figure 1).

To address these limitations, we extend procedural text comprehension with an additional task of predicting the dependencies between steps, in the form of their effects and which subsequent action(s) become possible. Building upon the state-of-the-art framework for predicting effects of actions, we present a new model, called XPAD (“eXplaining Action Dependencies”) that also considers the purpose of those effects. Specifically, XPAD biases those predictions towards those that (1) explain more of the actions in the paragraph and (2) are more plausible with respect to background knowledge. On a benchmark dataset for procedural text comprehension, ProPara (Dalvi et al., 2018), XPAD significantly improves on the prediction and explanation of action dependencies compared to prior systems, while also matching state-of-the-art results on the original tasks. We thus contribute:

1. A new task for procedural text comprehension, namely predicting and explaining the dependencies between actions (“what depends on what, and why”), including an additional dependency graph dataset for ProPara.
2. A model, XPAD, that significantly outperforms prior systems at predicting and explaining action dependencies, while maintaining its performance on the original tasks in ProPara.

2 Related Work

Understanding procedural text has a long history in AI. Early work attempted to construct semantic representations (“scripts”) of event sequences (or partial orders) from text, including representations of the goals, effects, and purpose of actions, e.g., (Schank and Abelson, 1977; DeJong, 1979; Cullingford, 1986; Mooney, 1986). Some of these systems could explain why actions occurred in text, similar to our goals here, but only on a handful of toy examples using hand-coded background knowledge, and proved difficult to scale. More recent work on event extraction and script learning has proved more effective at extracting event/action sequences from text using statistical methods, e.g., (Chambers and Jurafsky, 2008), and neural techniques, e.g., (Modi and Titov, 2014; Modi, 2016;

Pichotta and Mooney, 2016), but have largely focused on *what* happens and in what order, rather than *why*. For example, such representations cannot answer questions about which events would fail if an earlier action in a sequence did not occur. The 2014 ProRead system (Scaria et al., 2013; Berant et al., 2014) included dependency relationships between events that it extracted, but assessed dependencies based on surface language cues, hence could not explain why those dependencies held.

There has also been a line of research in reading procedural text in which the goal has been to track how entities’ states change with time, e.g., EntNet (Henaff et al., 2017), ProStruct (Tandon et al., 2018), and Neural Process Networks (Bosse-lut et al., 2018), applied to procedural text such as cooking recipes (Kiddon et al., 2016), science processes (Dalvi et al., 2018), or toy stories (Weston et al., 2015). Using annotated data, these systems learn to predict the effects of actions from text, allowing simple simulations of how the world changes throughout the process. We build on this line of work to identify the purpose of actions, by connecting their effects to subsequent actions.

In addition to signal from training data, many systems use background knowledge to help bias predictions towards those consistent with that knowledge. For example, (McLauchlan, 2004) predicts prepositional phrase attachments that are more consistent with unambiguous attachments derived from thesaurii; (Clark and Harrison, 2009) tune a parser to prefer bracketings consistent with frequent bracketings stored in a text-derived corpus of bracketings; and (Tandon et al., 2018) predicts state changes consistent with those seen in a large text corpus. For our purposes, while KBs such as (Speer and Havasi, 2013; Chu et al., 2017; Park and Nezhad, 2018) contain useful information about action dependencies (e.g., “smoking can cause cancer”), they lack explanations for those links. Instead, we create a KB of dependency links with explanations (Section 5.3), allowing us to similarly bias predictions with background knowledge.

3 Problem Definition

The input to our system is a paragraph of procedural text, along with a list of the *participant entities* in the procedure. The output is a *state change matrix* highlighting the changes undergone by the entities, as well as a *dependency explanation graph*

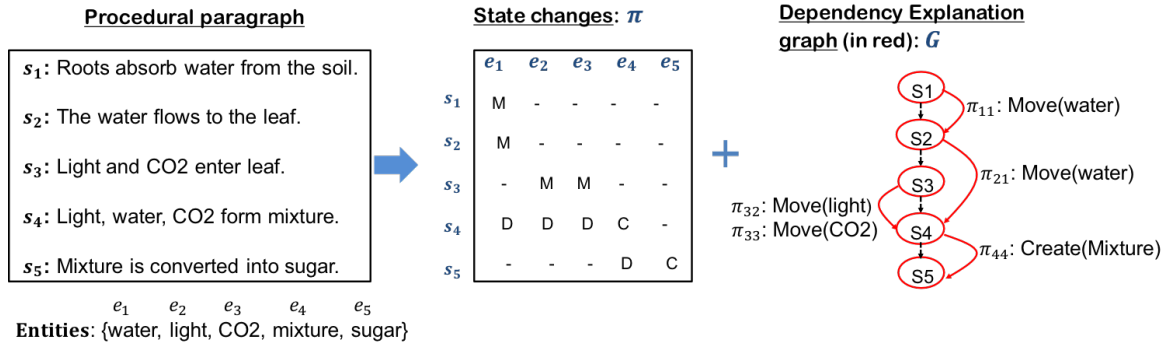


Figure 2: A procedural paragraph and target entities are provided as input. The task is to predict the state change matrix π , along with the dependency explanation graph \mathcal{G} between steps (red, overlaid on the step sequence). π_{ij} refers to the state change happening to entity e_j in step s_i . In the ProPara task, there are 4 possible state changes denoted by 'M' (Move), 'C' (Create), 'D' (Destroy), and '-' (no change). Further, \mathcal{G} describes which steps enable which other steps and how. For example, s_4 enables s_5 by creating mixture, required by s_5 .

that encodes “what action¹ depends on what, and why”, as illustrated in Figure 2. This problem definition is derived from the one used for the ProPara dataset (Dalvi et al., 2018), with the key addition of the dependency explanation graph. Although we use ProPara to illustrate and evaluate our work here, our approach is not specific to the details of that particular dataset.

Paragraph: We define a paragraph of procedural text $S = \langle s_1, \dots, s_T \rangle$ as a sequence of sentences that describe a procedure (e.g., photosynthesis). We assume that sentences are chronologically ordered and treat a sentence s_t as a step executed at time t .²

Participant Entities: We assume that the set of participant entities $E = \{e_1, \dots, e_n\}$ in the paragraph S is given. Notice that E includes only entities that participate in the process where their state changes at some steps.

State Change Matrix: For each entity $e_j \in E$, the system tracks how its properties change (e.g., location, existence) after each step. We enumerate all the state changes in a $T \times n$ matrix π , where π_{ij} denote how e_j has changed after step s_i . For ProPara, we use just four possible state changes for π_{ij} , namely {*Move*, *Create*, *Destroy*, *None*}, sufficient to model the properties tracked in the dataset. (More generally, additional state changes could be used provided their effects on entity properties is clearly defined.) *Move* means that the location of e_j changes after s_i . *Create* and *Destroy*

indicate e_j starts or ceases to exist. *None* means that the state of e_j remains unchanged. A state change can optionally take arguments, e.g., *Move* is associated with before and after locations.

Dependency Explanation Graph: In addition to being able to predict when entities undergo state changes, the system also needs to understand the dependency relationships between steps in a procedure. That is, for a step s_i to be executable, the system must be able to identify which previous steps are required to have been completed first. We represent these dependency relationships in procedural text using a *dependency explanation graph* $\mathcal{G} = \langle S, \mathcal{E} \rangle$, where each **node** is a step $s_i \in S$ and a **directed edge** $(s_i, s_j) \in \mathcal{E}$ indicates that s_i is a precondition of s_j ($i < j$). Moreover, each edge is associated with the **explanation** in the form of the entity state change that begets this dependency. For example, in the example shown in Fig. 2, both s_2 and s_3 are the parent nodes of s_4 . For *light*, *water*, *CO₂* to form the *mixture* in s_4 , the *water* has to move to the leaf in s_2 (i.e., $\pi_{21} = \text{Move}$), and *light* and *CO₂* also need to come to the leaf in s_3 (i.e., $\pi_{32} = \pi_{33} = \text{Move}$). Note that the dependency graph is not the “script” (event sequence) of the process, but an overlay on the script that explains *why* some actions need to happen before others. Note also that the state changes are fully specified (including the from and to locations) - the location information is dropped in Figure 2 for simplicity.

4 The Dependency Graph Dataset

Dependency graphs were added to the ProPara dataset using a mixture of manual and automated methods. First, an algorithm was used to estimate

¹We use a broad definition of action to mean any event that changes the state of the world (including non-volitional events such as roots absorbing water).

²This assumption holds in ProPara, the dataset used in this work. However, our techniques can generally be applied as long as a partial order of events is given.

dependencies using a heuristic method (below). Then, for the test set, these dependencies were manually reviewed and corrected. Approximately 15% of these dependencies needed to be changed, suggesting the heuristic algorithm is a reasonable approximation of the ground truth. The train and dev sets remain with the automatically generated (noisy but more numerous) dependency graphs.

As well as helping to augment the ProPara dataset, the algorithm can be used to automatically add approximate explanation graphs to existing models’ state change predictions. On first sight, this might seem like a simple solution to the dependency prediction task. However, as we show later, this approach does not produce good results. This is because models’ state change predictions were generated without regard for their explainability, and hence prediction errors can cascade into explainability errors. A better solution, which is the basis of XPAD, is to minimize a *joint* loss for both state change and explainability. In this way, an algorithm will be steered towards state changes that are also explainable (Figure 1).

The dependency graph algorithm is as follows, and is based on a *coherence assumption*: If step s_j changes the state of entity e_k , we assume that the reason s_j was included in the paragraph is because the next step mentioning e_k requires that change as a precondition. By searching forward for the first subsequent step s_j in which e_k is again mentioned, or changes state again, we add an enable edge in the dependency explanation graph that points from s_i to s_j , with the explanation label π_{ik} . As shown in Sec. 7.1, these dependency graphs that we added are mostly accurate ($F1 \approx 0.85$).

5 XPAD Model

Our model builds on the approach used in the ProStruct system (Tandon et al., 2018), namely an encoder-decoder approach with beam search decoding. We follow its design for the encoder, but use a modified decoder that also generates dependency graphs, and biases search towards graphs that are both more connected and more a priori likely.

5.1 Encoder

For each sentence s_t and each entity e_j , the encoder creates a vector c_{tj} , capturing how the actions in s_t affect e_j (Figure 3). During encoding, each word w_i in s_t is first represented by its GloVe vector, concatenated with two indicator variables: whether

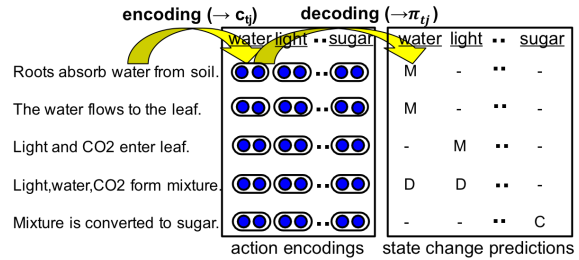


Figure 3: Encoder-decoder architecture used in XPAD: First encode the action of each sentence s_t on entity e_j , then decode to a predicted state change (state change arguments not shown). For example, the action *Roots absorb water from soil* is predicted to *Move* (M) the *water* (from soil to roots, not shown). Global predictions are generated through beam search.

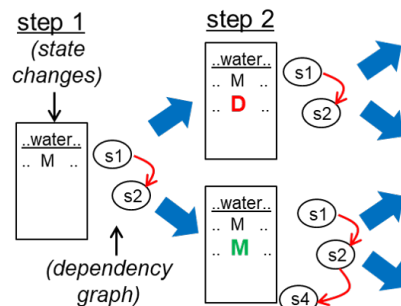


Figure 4: XPAD’s beam search for the best decoding, one sentence (step) at a time, showing the predicted state changes and the resulting dependency graph after each step. In step 2, XPAD chooses between predicting whether water is *destroyed* (D, upper figure) or *moved* (M, lower). Predicting M results in a more connected dependency graph (as water is mentioned again in step 4), hence the lower choice is likely preferred (Eq. (4)) by our *dependency graph score*. This score also assesses how a priori likely it is that s_4 ’s movement of water enables s_5 , using a background KB (Eq. (5)).

the word refers to e_j and whether the word is a verb. The sentence is then fed into a BiLSTM to generate a contextualized vector h_i , which is then passed to a bilinear attention layer: $a_i = h_i B h_{ev} + b$, where B and b are learned parameters, and h_{ev} is the concatenation of h_e and h_v (the averaged contextualized embedding of the entity and the verb words, respectively). The output vector c_{tj} is the attention-weighted sum of the h_i : $c_{tj} = \sum_{i=1}^l a_i \cdot h_i$.

Each c_{tj} will be decoded (Sec. 5.2) into one of K possible state changes. (For the ProPara application, $K = 4$ for {*create, move, destroy, none*}). As preparation for this, we pass each c_{tj} through a feedforward layer to generate K logits that denote how likely each entity e_j is to undergo each state change during step s_t . For changes that take addi-

tional arguments (e.g., *move*), the arguments are chosen using standard span prediction techniques over the encoded input h_i (Seo et al., 2017a).

5.2 Decoder

The goal of decoding is to determine the state change matrix π and the dependency graph \mathcal{G} . While each element π_{tj} of the state change matrix π could be determined solely by choosing the state change with the highest logit value, this greedy, *local* approach often results in nonsensical predictions (e.g., an entity is moved before being created). To avoid this, XPAD instead performs a beam search of possible decodings, one sentence s_t at a time, using a scoring function that includes terms that ensure *global* consistency, as shown in Fig. 4. Let π_t be the state change matrix up to step t , and \mathcal{G}_t be the dependency graph up to step t . As π_t is constructed, \mathcal{G}_t is derived from it deterministically using the same heuristic procedure described in Section 4. Most importantly, the scoring function used in the beam search is a function of *both* π_t and \mathcal{G}_t . This directs search towards state change predictions that are both globally consistent and produce a dependency graph that is well connected and a priori likely. Finally, the decoder outputs the complete state change grid π ($= \pi_T$) and dependency graph \mathcal{G} ($= \mathcal{G}_T$) from step 1 to T .

5.3 Dependency Aware Scoring Function

At each decoding step t , XPAD explores different options of π_t (state changes for all entities) and \mathcal{G}_t (corresponding dependency explanation graphs) as illustrated in Fig. 4. In particular, it scores the candidates at step t based on two components:

$$\phi(\pi_t, \mathcal{G}_t) = \lambda \cdot f(\pi_t) + (1 - \lambda) \cdot g(\mathcal{G}_t | \pi_t), \quad (1)$$

where (1) $f(\pi_t)$ is the **state change score** based on the likelihood of selected state changes at step t given the text and the state change history from steps s_1 to s_{t-1} , (2) $g(\mathcal{G}_t | \pi_t)$ is the **dependency graph score** based on the connectivity and likelihood of the resulting dependency explanation graph \mathcal{G}_t , and (3) λ is a hyper-parameter that determines the importance of the accuracy of the state changes vs. the coherence of the action dependency graph.

State Change Score: To compute $f(\pi_t)$, we reuse the scoring function proposed by ProStruct:

$$f(\pi_t) = \sum_{j=1}^{|E|} \left(\alpha \cdot \text{logit}(\pi_{tj}) + (1 - \alpha) \cdot \log P_{\text{ext}}(\pi_{tj} | e_j) \right), \quad (2)$$

where α is a hyper-parameter controlling the degree of bias, $\text{logit}(\pi_{tj})$ is the logit value supplied by the encoder for the local prediction, and $P_{\text{ext}}(\pi_{tj} | e_j)$ indicates how likely entity e_j will go through some change defined in π_{tj} , based on the topic of the procedural text. We use the knowledge base published by Tandon et al. (2018) to compute $P_{\text{ext}}(\pi_{tj} | e_j)$.

Dependency Graph Score: $g(\mathcal{G}_t | \pi_t)$ is the score of the resulting dependency graph given the selected state changes π_t , defined as:

$$g(\mathcal{G}_t | \pi_t) = \beta \cdot g_{\text{edge}}(\mathcal{G}_t | \pi_t) + (1 - \beta) \cdot g_{\text{kb}}(\mathcal{G}_t) \quad (3)$$

where (1) $g_{\text{edge}}(\mathcal{G}_t | \pi_t)$ scores π_t 's contributions to the dependency graph, (2) $g_{\text{kb}}(\mathcal{G}_t)$ encodes how *a priori* likely the added dependency edges, computed using background knowledge, and (3) β is a hyperparameter tuned on the dev set.

With the help of the **dependency graph score**, XPAD is able to bias the search toward predictions that have an identifiable purpose (i.e., dependency edges), and toward dependency graphs that are more likely according to the background knowledge. Next, we describe scores g_{edge} and g_{kb} in more details.

(a) Score g_{edge} : Actions with Purpose

We compute g_{edge} in Eq. (3) to bias the search toward dependency graphs that are more connected:

$$g_{\text{edge}}(\mathcal{G}_t | \pi_t) = \sum_{j=1}^{|E|} \log \left(1 + \text{score}_p(\mathcal{G}_t | \pi_{tj}) \right), \quad (4)$$

where score_p is assigned based on whether π_{tj} results in a purpose for step s_t or not, as follows:

$$\text{score}_p(\mathcal{G}_t | \pi_{tj}) = \begin{cases} +c, & \text{if } \pi_{tj} \text{ adds an edge to } \mathcal{G}_t; \\ 0, & \text{if no edge can be added} \\ & \text{to } \mathcal{G}_t \text{ as } e_j \text{ isn't mentioned later;} \\ -c, & \text{if } \pi_{tj} \text{ doesn't add an edge} \\ & \text{to } \mathcal{G}_t \text{ and } e_j \text{ is mentioned later.} \end{cases}$$

where $0 < c < 1$ is a hyper-parameter tuned on the dev set. Conceptually, this scoring function checks whether a particular state change prediction results in a connection in the dependency explanation graph \mathcal{G}_t . For example, if $\pi_{tj} = \text{Create entity } e$ and e is mentioned in a later step s_k , then an edge will be added between s_t and s_k to \mathcal{G}_t . Consequently, $\text{score}_p = c > 0$, resulting in a positive boost in the score for π_{tj} .

(b) Score g_{kb} : Action Dependency Priors

To distinguish likely dependency links (e.g., \mathcal{E}_{ij} in \mathcal{G}_t) from unlikely ones, we also bias the predictions in Equation 3 with g_{kb} . The scoring function g_{kb} computes the prior likelihood of each dependency link \mathcal{E}_{ij} that would be added to \mathcal{G}_t , as a result of the predicted actions $\pi_{ik} \in \pi_i$. This bias is particularly helpful when there is a limited amount of training data. If \mathcal{E}_t is the set of edges added to \mathcal{G}_t because of selected state changes π_t , then

$$g_{KB}(\mathcal{G}_t) = \sum_{\mathcal{E}_{ij} \in \mathcal{E}_t} \log(1 + score_e(\mathcal{E}_{ij})) \quad (5)$$

where $score_e(\mathcal{E}_{ij})$ scores the action dependency link (if any, 0 otherwise) added by action π_{ik} on an entity e_k that is common to steps s_i and s_j . For example, in Figure 2, action π_{11} adds an edge $MOVE(water)$, between sentences “*Roots absorb water.*” and “*The water flows to the leaf.*”

We need to estimate the likelihood of \mathcal{E}_{ij} (i.e., the likelihood that s_i can enable s_j via π_{ik}). This information (especially π_{ik}) is not present in any existing KB. Therefore, we train a model using a large collection of positive and negative examples of valid/invalid edges.

To generate these training examples, we first extract a large collection of procedural texts from WikiHow.com, which contains 1.75M steps from 227K processes across real-world domains, including health, finance, education, home, food, hobbies, etc. On each text, we apply a rule-based system (Clark et al., 2018) that noisily generates π_{train} , and then apply the heuristics from Section 4 to obtain \mathcal{G}_{train} . The distribution in \mathcal{G}_{train} can be quite different from that in ProPara (the current task), so we append \mathcal{G}_{train} with dependency graphs obtained from the training set in ProPara. We then decompose \mathcal{G}_{train} into its \mathcal{E}_{ij} edges (negative examples are created by reversing these edges). This leads to 324,462 training examples. We then add 2,201 examples derived from the ProPara train set. This use of hand-written rules to generate a large number of potentially noisy examples follows others in the literature, e.g. (Sharp et al., 2016; Ahn et al., 2016; Bosselut et al., 2018).

To accommodate for lexical variations, we embed the database of training data in a neural model. Our model itself takes as input \mathcal{E}_{ij} and outputs the likelihood of \mathcal{E}_{ij} . To do this, an embedding for \mathcal{E}_{ij} is created using a deep network of biLSTMs, producing a contextual embedding based on the token

level embeddings of s_i , s_j and the state change vector π_{ik} . This contextual embedding is then decoded using a feedforward network to predict a score for whether s_i enabled s_j through π_{ik} . The loss function is designed such that errors on the training examples coming from ProPara are penalized θ times more than those from WikiHow, where θ is a hyperparameter tuned on the dev set.

While g_{edge} only scores whether the same entity is used in the two connected events or not, ignoring location information, g_{kb} models how likely there is a dependency between the two events, including using the from/to location of an entity. For example, g_{kb} can model that moving ‘light’ and ‘CO2’ to ‘location=leaf’ is important before they can be turned into a ‘mixture’ (Figure 2).

5.4 Training and Testing XPAD

At training time, XPAD follows only the correct (gold) path through the search space, and learns to minimize the joint loss of predicting the correct state changes and dependency explanation graph for the paragraph. At test time, XPAD performs a beam search to predict the most likely state changes and dependency explanation graph.

5.5 Implementation Details for XPAD

We implement XPAD in PyTorch using AllenNLP (Gardner et al., 2018). We use the dataset reader published in ProStruct’s publicly available code. We use 100D Glove embeddings (Pennington et al., 2014), trained on Wikipedia 2014 and Gigaword 5 corpora (6B tokens, 400K vocab, uncased). Starting from glove embeddings appended by entity and verb indicators, we use bidirectional LSTM layer to create contextual representation for every word. We use 100D hidden representations for the bidirectional LSTM (Hochreiter and Schmidhuber, 1997) shared between all inputs (each direction uses 50D hidden vectors). The attention layer on top of BiLSTM, uses a bilinear similarity function similar to (Chen et al., 2016) to compute attention weights over the contextual embedding for every word.

To compute the likelihood of all state changes individually, we use a single layer feedforward network with input dimension of 100 and output 4. We then use constrained decoder during training as explained in Section 5.2. We tune the hyperparameters in Equations 1, 2, and 3 as $\lambda = 0.5$, $\alpha = 0.8$, $\beta = 0.8$. We use Adadelat optimizer (Zeiler, 2012) with learning rate 0.2 to minimize total loss.

To make predictions, we use a constrained decoder with beam size of 20, i.e. top 20 choices are explored at each step.

6 Experiments

6.1 Evaluation metric and baselines

To evaluate XPAD, we measure its performance on dependency explanations, as well as its performance on the original state change prediction task. For state change prediction, we use the same dataset and evaluation metric described in (Tandon et al., 2018), consisting of 1095 test questions for the inputs, outputs, conversions, and movements in the process (straightforwardly derivable from the predicted state change matrix). For dependency explanations, we compare the curated, gold graphs \mathcal{G}_{gold} that we added to ProPara (Section 4) with the predicted dependency graphs \mathcal{G}_{pred} . Each edge in a graph contains 3 elements of explanation for “Why s_t ?”, namely “ s_t enables s_u due to a state change π_{tk} in entity e_k ”, resulting in 2,814 elements to predict in the test set. We compute the precision and recall of predicting these elements for all steps s_t , yielding an overall F_1 score.

We consider the state-of-the-art process comprehension models reported in (Tandon et al., 2018) as our baselines, including Recurrent Entity Networks (EntNet) (Henaff et al., 2017), Query Reduction Networks (QRN) (Seo et al., 2017b), ProLocal and ProGlobal (Dalvi et al., 2018) and finally, ProStruct (Tandon et al., 2018). These models output state-change predictions, which are used to create the corresponding dependency explanation graph using the method in Section 4, but do not bias their state change predictions towards those that appear purposeful.

6.2 Results

Results on the proposed dependency task:

Model	P	R	F ₁
ProLocal	24.7	18.0	20.8
QRN	32.6	30.3	31.4
EntNet	32.8	38.6	35.5
ProStruct	76.3	21.3	33.4
ProGlobal	43.4	37.0	39.9
XPAD	62.0	32.9	43.0

Table 1: Results on the **dependency** task (test set).

Table 1 reports results of all models on the new **dependency** task. XPAD significantly outperforms

the strongest baselines, ProGlobal and ProStruct, by more than 3 points F_1 . XPAD has much higher precision than ProGlobal with similar recall, suggesting that XPAD’s dependency-aware decoder helps it select more accurate dependencies. Compared with ProStruct, it yields more than 11.6 points improvement on recall. As XPAD adds a novel dependency layer on top of the ProStruct architecture, we note that all these gains come exclusively from the dependency layer.

Impact of XPAD components:

Table 2 shows the impact of removing dependency graph scores from XPAD. Removing g_{kb} , results in a substantial (9 points) drop in recall, since the action dependency priors uses background knowledge to help the model discover dependency links that the model could not infer from the current paragraph. Further, removing g_{edge} score results in around 2 points drop in recall and a substantial increase in precision (F_1 is still lower). This is because the g_{edge} score encourages the model to predict state changes that result in more action dependency links, often at the cost of a drop in precision. Thus, the g_{edge} and g_{kb} scores together help XPAD discover meaningful action dependency links.

Ablation	Dependency Task		
	P	R	F ₁
XPAD	62.0	32.9	43.0
- g_{kb}	67.9	23.1	34.5
- g_{kb} - g_{edge}	76.3	21.3	33.4

Table 2: Effect of ablating g_{edge} and g_{kb} from XPAD (test set).

Results on the previous state-change task:

On the original state-change prediction task, we find that XPAD performs slightly better (by 0.7 points F_1) than the best published state-of-the-art system ProStruct³, even though Equation 1 is not optimized solely for that task (Table 3). This illustrates that encouraging purposefulness in action

³Since XPAD was developed, two higher unpublished results of 57.6 and 62.5 on the state-change task have appeared on arXiv (Das et al., 2019; Gupta and Durrett, 2019), their systems developed contemporaneously with XPAD. In principle XPAD’s approach of jointly learning both state changes and dependencies could be also applied in these new systems. Our main contribution is to show that jointly learning state changes and dependencies can produce more rational (explainable) results, without damaging (here, slightly improving) the state change predictions themselves.

prediction not only produces more explainable action sequences, but can improve the accuracy of those action predictions themselves.

	P	R	F ₁
ProStruct	74.3	43.0	54.5
XPAD	70.5	45.3	55.2

Table 3: Results on the **state-change** task (test set), comparing with the best published prior result.

7 Analysis and Discussion

7.1 Dependency Graph Computation Errors

Our heuristic algorithm for estimating the dependency graph from state change predictions (Section 4) makes a *coherence assumption*, namely that the purpose of a state change on entity e is to enable the next event mentioning e . We now describe classes of errors that the algorithm makes, by summarizing errors that had to be corrected when curating the gold explanation graphs for the test set.

1. Coreference (synonymy) (35% of errors): Reference to a changed entity is missed if an alternative wording is used, e.g.,

snow becomes ice...the mass grows smaller...

2. Unstated Linkage (25%): Sometimes use of an entity will be implicit, e.g.,

Animals eat plants...Animals make waste [from the plants]...

preventing finding what “eat plants” enables.

3. Ellipsis (15%): (Sometimes ungrammatical) elided references to earlier entities will hide the purpose of those entity’s state changes, e.g.,

Water flows downwards... Enters the dam...

Thus, this analysis suggests several ways that the dependency graph computation could be improved.

4. Bridging Anaphora (15%): In some cases, a sentence will refer indirectly to a changed entity, e.g., by referring to its part, whole, or some other association (i.e., an associative/bridging anaphoric reference (Wei, 2014)). As the algorithm does not resolve such indirect references, the purpose of the action changing that entity is unrecognized, e.g.,

spark plug causes a spark...explosion occurs...

The leaf absorbs CO₂... The plant produces O₂...

5. Bad State Change Annotations (5%): In a few cases, state change annotations are missing or in error in the gold dataset, resulting in missing or incorrect prediction of their purpose.

6. Long-Range Dependencies (5%): We assume that an entity’s change is to enable the next men-

tioned use of that entity. This assumption is based solely on typical protocols of discourse, and occasionally is violated. For example:

Rainwater picks up CO₂... Rainwater goes into the soil... The water dissolves limestone...

Here the first action is a prerequisite for the third, not the second, thus violating our assumption and creating an error in the dependency graph.

Note that around 45% of the sentences in ProPara have no associated state change because of unmodeled state changes (other than create/destroy/move) or the presence of stative sentences (no resultant state changes). As a result, the purpose of such sentences is inherently unrecoverable given these annotations, and out of reach of XPAD (and any other model).

7.2 Qualitative Analysis of XPAD’s Results

Corrected Predictions

In many cases, the bias toward enables links results in XPAD predicting a state change when previously *none* had been predicted. For example, in sentence (6) of the paragraph snippet below:

(6) *Millions of years later the fossil forms.*

(7) *A person finds the fossil.*

PROSTRUCT (for step 6): *None*

XPAD: *CREATE(fossil)* [correct]

PROSTRUCT incorrectly predicted no state changes from (6), while XPAD (correctly) predicted that a fossil was created. The extra evidence causing XPAD to make this prediction is that it adds an edge in the dependency graph, giving step (6) a purpose, namely to enable step (7) in which the fossil is found. There are many such examples.

Erroneous Predictions

Both of XPAD’s new biases encourage XPAD to make state-change predictions that result in more enables edges, either by counting them (g_{edge}) or summing their likelihoods (g_{kb}), resulting in some overlap in their overall effect (Table 6.2). The bias can help predict missing state changes (e.g., above), but can also cause XPAD to “hallucinate” state changes with weak evidence, simply to give each sentence a purpose. An example is:

(5) *Carbon-based mixture stays underground.*

(6) *Humans discover this carbon-based mixture.*

PROSTRUCT (for step 5): *None* [correct]

XPAD: *CREATE(carbon-based mixture)*

Here, there is no action associated with step (5). However, in part because this is the first mention of carbon-based mixture, XPAD incorrectly labels

it as being created, giving step (5) a purpose of enabling step (6). Although such errors occur, they are less common than the corrected predictions, hence the improved overall performance.

Comparing the ablated versions of XPAD, we can also see how the different biases play out. Without g_{kb} , XPAD can sometimes predict enables edges that are nonsensical with respect to background knowledge, for example,

(1) *Nitrogen exists naturally in the atmosphere.*

(9) *Bacteria turn nitrogen into a gas.*

XPAD (no g_{kb}) (for step 1): CREATE (gas)

XPAD: None [correct]

In this example, XPAD without g_{kb} produces an incorrect prediction for (1) (that gas has been created), as it results in an enables link from step (1) to (9). However, the background KB deems it unlikely that (1) enables (9); as a result, the full XPAD makes no such prediction.

8 Conclusion

Our goal is to better comprehend procedural text by not only predicting *what* happens, but *why*. To do this, we have expanded the traditional state-tracking task with the additional task of predicting and explaining dependencies between steps, and presented a new model, XPAD, for these tasks. XPAD is biased to prefer predictions that have an identifiable purpose (enables a subsequent step), and where that purpose is more plausible (judged using a large corpus). Experiments show that XPAD significantly improves the predictions of correct dependencies between actions, while matching state-of-the-art results on the earlier tasks.

Although our experiments have been with ProPara, there is nothing intrinsic to XPAD’s architecture that limits it to that dataset’s design: Given appropriate annotations, XPAD could be trained with a richer set of state change operators, longer paragraphs, and (with the addition of a temporal ordering module, e.g., (Ning et al., 2017)) non-chronological event orderings. In addition, given additional hand-labeled training data, a system might learn to directly predict dependency links (instead of deriving them heuristically from state changes). These would be valuable new directions to pursue. The Dependency Graph dataset is available within <http://data.allenai.org/propara>

Acknowledgements

We are grateful to the AllenNLP and Beaker teams at AI2, and for the insightful discussions with other Aristo team members. Computations on beaker.org were supported in part by credits from Google Cloud.

References

- Sungjin Ahn, Heeyoul Choi, Tanel Pärnamaa, and Yoshua Bengio. 2016. A neural knowledge language model. *arXiv preprint arXiv:1608.00318*.
- Jonathan Berant, Vivek Srikumar, Pei-Chun Chen, Abby Vander Linden, Brittany Harding, Brad Huang, Peter Clark, and Christopher D Manning. 2014. Modeling biological processes for reading comprehension. In *Proc. EMNLP’14*.
- Antoine Bosselut, Omer Levy, Ari Holtzman, Corin Ennis, Dieter Fox, and Yejin Choi. 2018. Simulating action dynamics with neural process networks. *6th International Conference on Learning Representations (ICLR)*.
- Nathanael Chambers and Daniel Jurafsky. 2008. Unsupervised learning of narrative event chains. In *Proc. ACL’08*.
- Danqi Chen, Jason Bolton, and Christopher D. Manning. 2016. A thorough examination of the cnn/daily mail reading comprehension task. *CoRR*, abs/1606.02858.
- Cuong Xuan Chu, Niket Tandon, and Gerhard Weikum. 2017. Distilling task knowledge from how-to communities. In *WWW*.
- Peter Clark and Philip Harrison. 2009. Large-scale extraction and use of knowledge from text. In *K-CAP*.
- Peter Clark, Bhavana Dalvi Mishra, and Niket Tandon. 2018. What happened? leveraging verbnet to predict the effects of actions in procedural text. *arXiv preprint arXiv:1804.05435*.
- Richard Cullingford. 1986. The script applier mechanism. In *Readings in natural language processing*, pages 627–649. Morgan Kaufmann Publishers Inc.
- Bhavana Dalvi, Lifu Huang, Niket Tandon, Wen-tau Yih, and Peter Clark. 2018. Tracking state changes in procedural text: A challenge dataset and models for process paragraph comprehension. In *NAACL-HLT’18*.
- Rajarshi Das, Tsendsuren Munkhdalai, Xingdi Yuan, Adam Trischler, and Andrew McCallum. 2019. Building dynamic knowledge graphs from text using machine reading comprehension. *arXiv preprint arXiv:1810.05682*. ICLR 2019.

- Gerald DeJong. 1979. Prediction and substantiation: A new approach to natural language processing. *Cognitive Science*, 3(3):251–273.
- Matt Gardner, Joel Grus, Mark Neumann, Oyvind Tafjord, Pradeep Dasigi, Nelson Liu, Matthew Peters, Michael Schmitz, and Luke Zettlemoyer. 2018. Allennlp: A deep semantic natural language processing platform. *arXiv preprint arXiv:1803.07640*.
- Aditya Gupta and Greg Durrett. 2019. Tracking discrete and continuous entity state for process understanding. *arXiv preprint arXiv:1904.03518*. (To appear in NAACL’19 workshop on Structured Prediction for NLP).
- Mikael Henaff, Jason Weston, Arthur Szlam, Antoine Bordes, and Yann LeCun. 2017. Tracking the world state with recurrent entity networks. In *ICLR*.
- Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation*, 9(8):1735–1780.
- Chloé Kiddon, Luke S. Zettlemoyer, and Yejin Choi. 2016. Globally coherent text generation with neural checklist models. In *EMNLP*.
- Mark McLauchlan. 2004. Thesauruses for prepositional phrase attachment. In *CoNLL*.
- Ashutosh Modi. 2016. Event embeddings for semantic script modeling. In *CoNLL*.
- Ashutosh Modi and Ivan Titov. 2014. Inducing neural models of script knowledge. In *CoNLL*.
- Raymond J Mooney. 1986. Generalizing explanations of narratives into schemata. In *Machine Learning*, pages 207–212. Springer.
- Qiang Ning, Zhili Feng, and Dan Roth. 2017. A structured learning approach to temporal relation extraction. In *EMNLP*, pages 1027–1037.
- Hogun Park and Hamid R. Motahari Nezhad. 2018. Learning procedures from text: Codifying how-to procedures in deep neural networks. In *WWW*.
- Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543.
- Karl Pichotta and Raymond J. Mooney. 2016. Using sentence-level lstm language models for script inference. In *ACL*.
- Aju Thalappillil Scaria, Jonathan Berant, Mengqiu Wang, Peter Clark, Justin Lewis, Brittany Harding, and Christopher D. Manning. 2013. Learning biological processes with global constraints. In *EMNLP*.
- Roger C Schank and Robert P Abelson. 1977. *Scripts, plans, goals, and understanding: An inquiry into human knowledge structures*. Psychology Press.
- Minjoon Seo, Aniruddha Kembhavi, Ali Farhadi, and Hannaneh Hajishirzi. 2017a. Bidirectional attention flow for machine comprehension. In *Proc. ICLR’17*.
- Minjoon Seo, Sewon Min, Ali Farhadi, and Hannaneh Hajishirzi. 2017b. Query-reduction networks for question answering. In *ICLR*.
- Rebecca Sharp, Mihai Surdeanu, Peter Jansen, Peter Clark, and Michael Hammond. 2016. Creating causal embeddings for question answering with minimal supervision. In *EMNLP’16*.
- Robyn Speer and Catherine Havasi. 2013. Conceptnet 5: A large semantic network for relational knowledge. In *The People’s Web Meets NLP*.
- Niket Tandon, Bhavana Dalvi Mishra, Joel Grus, Wentau Yih, Antoine Bosselut, and Peter Clark. 2018. Reasoning about actions and state changes by injecting commonsense knowledge. *EMNLP’18*, arXiv preprint arXiv:1808.10012.
- Zhao Wei. 2014. A survey of studies of bridging anaphora. *Canadian Social Science*, 10(3).
- Jason Weston, Antoine Bordes, Sumit Chopra, Alexander M Rush, Bart van Merriënboer, Armand Joulin, and Tomas Mikolov. 2015. Towards AI-complete question answering: A set of prerequisite toy tasks. *arXiv preprint arXiv:1502.05698*.
- Matthew Zeiler. 2012. ADADELTA: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*.