Reasoning about Actions and State Changes by Injecting Commonsense Knowledge

Niket Tandon*, **Bhavana Dalvi Mishra***, **Joel Grus**,
Wen-tau Yih, **Antoine Bosselut**, **Peter Clark**
Allen Institute for AI, Seattle, WA
{nikett,bhavanad,joelg,scottyih,antoineb,peterc}@allenai.org

Abstract

Comprehending procedural text, e.g., a paragraph describing photosynthesis, requires modeling actions and the state changes they produce, so that questions about entities at different timepoints can be answered. Although several recent systems have shown impressive progress in this task, their predictions can be globally inconsistent or highly improbable. In this paper, we show how the predicted effects of actions in the context of a paragraph can be improved in two ways: (1) by incorporating global, commonsense constraints (e.g., a non-existent entity cannot be destroyed), and (2) by biasing reading with preferences from large-scale corpora (e.g., trees rarely move). Unlike earlier methods, we treat the problem as a neural structured prediction task, allowing hard and soft constraints to steer the model away from unlikely predictions. We show that the new model significantly outperforms earlier systems on a benchmark dataset for procedural text comprehension (+8% relative gain), and that it also avoids some of the nonsensical predictions that earlier systems make.

1 Introduction

Procedural text is ubiquitous (e.g., scientific protocols, news articles, how-to guides, recipes), but is challenging to comprehend because of the dynamic nature of the world being described. Comprehending such text requires a model of the actions described in the text and the state changes they produce, so that questions about the states of entities at different timepoints can be answered (Bosselut et al., 2018).

Despite these challenges, substantial progress has been made recently in this task. Recent work – such as EntNet (Henaff et al., 2017), QRN (Seo et al., 2017b), ProLocal/ProGlobal (Dalvi et al., 2018), and NPN (Bosselut et al., 2018) – has focused on learning to predict individual entity states at various points in the text, thereby approximating the underlying dynamics of the world. However, while these models can learn to make local predictions with fair accuracy, their results are often globally unlikely or inconsistent. For example, in Figure 1, the neural ProGlobal model from Dalvi et al. (2018) learns to predict the impossible action of an object moving from itself (1), and the unlikely action of a turbine changing location (2). We observe similar mistakes in other neural models, indicating that these models have little notion of global consistency. Unsurprisingly, mistakes in local predictions compound as the process becomes longer, further reducing the plausibility of the overall result.

Procedural Text:

How hydroelectric electricity is generated:
1 Water flows downwards thanks to gravity.
2 The moving water spins the turbines in the power plant.
3 The turbines turn the generators.
4 The generators spin, and produce electricity.

Prior Neural Model's Predictions:

(1) water **moves** from the **water** to **gravity**
↓
(2) the turbine moves
from the water to the power plant
↓
(4) electricity is created at the generator

Figure 1: Poor predictions (in red) made by a prior neural model (ProGlobal) applied to an (abbreviated) paragraph from the ProPara dataset. ProGlobal predicts entity locations at each sentence, but the implied movements violate commonsense constraints (e.g., an object cannot move from itself (1)) and corpus-based preferences (e.g., it is rare to see turbines move (2)).

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

Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, pages 57–66
Brussels, Belgium, October 31 - November 4, 2018.

To address this challenge, we treat process comprehension as a *structured prediction task* and apply hard and soft constraints during reading. During training, our model, called ProStruct, learns to search for the most likely action sequence that is consistent with global constraints (e.g., entities cannot be destroyed after they have already been destroyed) and priors from background knowledge (e.g., turbines rarely change location). The model is trained end-to-end, with gradients backpropagating through the search path. We find that this approach significantly outperforms existing approaches on a benchmark dataset for process comprehension, mainly by avoiding the nonsensical predictions that earlier systems make.

Our contributions are twofold. First, we reformulate procedural text comprehension in a novel way: as a (neural) structured prediction task. This lets hard and soft constraints steer the model away from unlikely and nonsensical predictions. Second, we present a novel, end-to-end model that integrates these constraints and achieves state-of-the-art performance on an existing process comprehension dataset (Dalvi et al., 2018).

2 Related Work

Our work builds off a recent body of work that focuses on using neural networks to explicitly track the states of entities while reading long texts. These works have focused on answering simple commonsense questions (Henaff et al., 2017), tracking entity states in scientific processes (Dalvi et al., 2018; Clark et al., 2018), tracking ingredients in cooking recipes (Bosselut et al., 2018), and tracking the emotional reactions and motivations of characters in simple stories (Rashkin et al., 2018). Our work extends these methods and addresses their most common issues by using background knowledge about entities to prune the set of state changes they can experience as the model reads new text.

Prior to these neural approaches, some earlier systems for process comprehension did make use of world knowledge, and motivated this work. Like us, the system ProRead (Berant et al., 2014; Scaria et al., 2013) also treated process comprehension as structure prediction, using an Integer Linear Programming (ILP) formalism to enforce global constraints (e.g., if the result of event1 is the agent of event2, then event1 must enable event2). Similarly, Kiddon et al. (2015) used corpus-based priors to guide extraction of an "action graph" from recipes. Our work here can viewed as incorporating these approaches within the neural paradigm.

Neural methods for structure prediction have been used extensively in other areas of NLP, and we leverage these methods here. In particular we use a neural encoder-decoder architecture with beam search decoding, representative of several current state-of-the-art systems (Bahdanau et al., 2014; Wiseman and Rush, 2016; Vinyals et al., 2015). As our model's only supervision signal comes from the final prediction (of state changes), our work is similar to previous work in semantic parsing that extracts structured outputs from text with no intermediate supervision (Krishnamurthy et al., 2017).

State tracking also appears in other areas of AI, such as dialog. A typical dialog state tracking task (e.g., the DSTC competitions) involves gradually uncovering the user's state (e.g., their constraints, preferences, and goals for booking a restaurant), until an answer can be provided. Although this context is somewhat different (the primary goal being state discovery from weak dialog evidence), state tracking techniques originally designed for procedural text have been successfully applied in this context also (Liu and Perez, 2017).

Finally, our model learns to search over the best candidate structures using hard constraints and soft KB priors. Previous work in Neural Machine Translation (NMT) has used sets of example-specific lexical constraints in beam search decoding to only produce translations that satisfy every constraint in the set (Hokamp and Liu, 2017). In contrast, our work uses a set of global example-free constraints to prune the set of possible paths the search algorithm can explore. Simultaneously, a recent body of work has explored encoding soft constraints as an additional loss term in the training objective for dialogue (Wen et al., 2015), machine translation (Tu et al., 2016), and recipe generation (Kiddon et al., 2016). Our work instead uses soft constraints to re-rank candidate structures and is not directly encoded in the loss function.

3 Problem Definition

We first define the general task that we are addressing, before presenting our approach.

3.1 General Formulation

We define the task as follows. **Given:**

- A **paragraph of procedural text** S = an ordered set of sentences $\{s_1, ..., s_T\}$ describing

		Participants:		
Paragraph (seq. of steps):		water	turbine	electricity
	state0	?	power plant	-
Water flows downwards thanks to gravity				
	state1	?	power plant	-
The moving water spins the turbines in the power plant				
	state2	turbine	power plant	-
The turbines turn generators				
	state3	turbine	power plant	-
The generators spin, and produce electricity				
	state4	turbine	power plant	generator

Time

Figure 2: How the (simplified) paragraph in Figure 1 is annotated in ProPara. Each filled row shows the location of entities between each step ("?" denotes "unknown", "-" denotes "does not exist"). For example, in the last line (state4), the water is at the turbine.

a sequence of **actions**[1] about a given **topic** (a word or phrase).

- A **set of entities** $E = \{e_j\}$ representing the entities mentioned in the procedure or process. Each entity e_j is denoted by the set of its mentions in the paragraph, e.g., {`leaf, leaves`}
- A **set of properties** $P = \{p_k\}$ of entities to be tracked (e.g., location, existence)

predict:

- The **state** of each entity e_j after each sentence s_k, where an entity's state is the values of all its properties $\{p_k\}$. For example, in Figure 2, the state of the water after step 2 is {location(water) = turbine; exists(water) = true}.

This task definition covers the tasks used in earlier procedural text comprehension datasets. In bAbI tasks 1-3, a single propert (location) was tracked for a single entity throughout a paragraph (Weston et al., 2015). In the state tracking task of Bosselut et al. (2018), six properties (temperature, shape, etc.) were tracked for each ingredient in the recipe.

3.2 Data

In our work, we use the ProPara dataset (Dalvi et al., 2018) for both illustration and evalution. ProPara contains 488 paragraphs (3100 sentences) of a particular genre of procedural text, namely science processes (e.g., how hydroelectricity is generated). The dataset tracks two entity properties, existence and location, for all entities involved in each process, resulting in 81,000 annotations in the dataset. Figure 2 gives a (simplified) example of the data, visualized as an (entity x sentence) grid, where each column tracks a different entity (time progressing vertically downwards), and each row denotes the entities' state (existence and location) after each sentence. To evaluate the predictions, a set of templated questions whose answers can be computed from the predictions is posed (e.g., "What was destroyed, when and where?").

[1]We 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).

4 Model

We now describe our model, called ProStruct.

4.1 Overview

We approach the task by predicting the state *changes* that occur at each step of the text, using a vocabulary (size K) of the possible state change types that can occur given the domain and properties being modeled. For example, for the ProPara dataset, we model $K = 4$ types of state change: `move`, `create`, `destroy`, and `none`. `move` changes an entity's location from one place to another, `create` from non-existence to a location, and `destroy` from a location to non-existence. State changes can be parameterized by text spans in the paragraph, e.g., `move` takes a before and after location parameter. If a parameterized state change is predicted, then the model also must predict its parameter values from the paragraph.

Previous models for process comprehension make a sequence of local predictions about the entities' states, one sentence at a time, maintaining a (typically neural) state at each sentence. However, none have the ability to reverse earlier predictions should an inconsistency arise later in the sequence. ProStruct overcomes this limitation by reformulating the task as structured prediction. To do this, it uses a neural encoder-decoder from the semantic parsing literature (Krishnamurthy et al., 2017; Yin and Neubig, 2017) combined with a search procedure that integrates soft and hard constraints for finding the best candidate structure.

For each sentence and entity, the encoder first uses a bidirectional LSTM to encode the sentence and indicator variables identifying which entity is currently being considered (Figure 3). It then produces a (distributed) representation of the action that the sentence describes as being applied to that entity. During decoding, the model decodes each action embedding into a distribution over possible state changes that might result, then performs

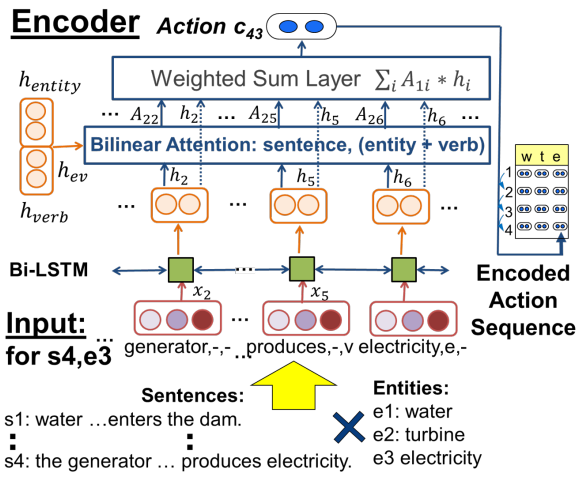

Figure 3: The encoder, illustrated for the ProPara domain with the paragraph from Figure 1. During encoding, ProStruct creates an action embedding c_{tj} representing the action at step t on entity e_k, for all entities at all steps. The overall action sequence (right-hand box) is the collection of these embeddings, for each entity (listed horizontally) and each step (listed vertically downwards).

a search over the space of possible state change sequences. Each node in the space is a partial sequence of state changes, and each edge is a prediction of the next state changes to add to the sequence (Figure 4).

During training, the model only follows the path along the gold sequence, and optimizes a loss function that drives up the likelihood of predictions along that path (thus driving down the probabilities for alternative, incorrect paths). At test time, the model does not have access to the gold path, and instead performs a beam search of the space to find the best candidate sequence.

Most importantly, by mapping the state change prediction problem to structured prediction, we can perform a search over the set of candidate paths that allows us to introduce hard and soft constraints that capture commonsense knowledge. Hard constraints are used to prune the search space (Equation 4 later), and soft constraints bias the search away from unlikely state changes via an additional term in the scoring function (Equations 5 and 6).

4.2 Encoder

The encoder operates over every $(s_t, e_j) \in S \times E$ pair to create an encoded representation c_{tj} of the action described by sentence s_t, as applied to entity e_j. In other words, we can consider the overall action to be represented by $|E|$ embeddings, one for each of the entities in E, encoding the action's effects on each. This novel feature allows us to model different effects on different entities by the same action. For example, a conversion action may simultaneously destroy one entity and create another. Figure 3 shows the encoder operating on s_4: "The generator spins, and produces electricity" and e_3: `electricity` from Figure 1.

Without loss of generality, we define an arbitrary sentence in S as $s_t = \{w_0, ..., w_I\}$. Each word w_i in the input sentence is encoded as a vector $x_i = [v_w : v_e : v_v]$, which is the concatenation of a pre-trained word embedding v_w for w_i, an indicator variable v_e for whether w_i is a reference to the specified entity e_j, and an indicator variable v_v for whether w_i is a verb. We use GloVe vectors as pre-trained embeddings (Pennington et al., 2014) and a POS tagger to extract verbs (Spacy, 2018).

Then, a BiLSTM is used to encode the word representations extracted above, yielding a contextualized vector h_i for each embedded word x_i that is the concatenated output of the backward and forward hidden states produced by the BiLSTM for word w_i. An attention over the contextualized embeddings h_i is performed to predict a distribution of weights over the sentence:

$$a_i = h_i * B * h_{ev} + b \quad (1)$$

$$c_{tj} = \sum_{i=1}^{I} a_i * h_i \quad (2)$$

where a_i is the attention weight for each contextualized embedding, c_{tj} is the vector encoding the action for the sentence-entity pair (s_t, e_j), B and b are learned parameters, and h_{ev} is the concatenation of the contextual embeddings of the hidden states where the entity h_e and verb h_v are mentioned:

$$h_{ev} = [\mu(\{h_i : x_i[v_e] = 1\}); \mu(\{h_i : x_i[v_v] = 1\}] \quad (3)$$

where μ is an average function, and $x_i[v_e]$ and $x_i[v_v]$ correspond to the entity indicator and verb indicator variables defined above for any word w_i, respectively. The output vector c_{tj} encodes the action at step s_t on entity e_j. This vector is computed for all steps and entities, populating a grid of the actions on each entity at each step (Figure 3).

4.3 Decoder

To decode the action vectors c_{tj} into their resulting state changes they imply, each is passed through a

feedforward layer to generate $logit(\pi_t^j)$, a set of logistic activations over the K possible state changes π_t^j for entity e_j in sentence s_t. (For ProPara, there are $K = 4$ possible state changes: `move`, `create`, `destroy`, `none`). These logits denote how likely each state change π_t^j is for entity e_j at sentence s_t. The decoder then explores the search space of possible state change sequences for the **whole** paragraph (Figure 4), using these likelihoods to score each visited sequence (Equation 6).

Let π_t be the set of state changes for *all* entities at time t, i.e., $\pi_t = \{\pi_t^j\}_{j=1..|E|}$, and let Π_t be the sequence of state changes from time 1 to t, i.e., $\Pi_t = [\pi_1, ..., \pi_t]$. Each node in the search space is a Π_t, and each edge adds a π_{t+1} to it so that it becomes Π_{t+1}:

$$\Pi_t \xrightarrow{\pi_{t+1}} \Pi_{t+1}$$

Given there are K possible values for π_t^j, the number of possible configurations for π_t at time t (i.e., the branching factor during search) is exponential: $K^{|E|}$, where $|E|$ is the number of entities in the paragraph.

To explore this exponential number of paths, after every sentence s_t, we prune branches $\Pi_t \rightarrow \Pi_{t+1}$ where Π_{t+1} is impossible according to background knowledge (described in Section 5.1). We define the boolean function over state change sequences:

$$\begin{aligned} \text{allowable}(\Pi) &= 1 \text{ if hard constraints satisfied} \\ &= 0 \text{ otherwise} \end{aligned} \quad (4)$$

and prune paths Π_{t+1} where $allowable(\Pi_{t+1}) = 0$. For example for ProPara, a state transition such as `DESTROY` $\rightarrow$ `MOVE` is not allowed because a hard constraint prohibits non-existent entities from being moved (Section 5.1).

While hard constraints remove impossible state change predictions, there may also be other state changes that are implausible with respect to background knowledge. For example, commonsense dictates that it is unlikely (but not impossible) for plants to be destroyed during photosynthesis. Accordingly, our inference procedure should discourage (but not prohibit) predicting plant destruction when reading about photosynthesis. To discourage unlikely state changes, we make use of soft constraints that estimate the likelihood of a particular state change associated with an entity, denoted as:

$$P(\pi^j | e_j, topic) \quad (5)$$

In Section 5.2, we describe how these likelihoods can be estimated from large-scale corpora. We add this bias as an additional term (the second term below) when scoring the addition of π_{t+1} to the

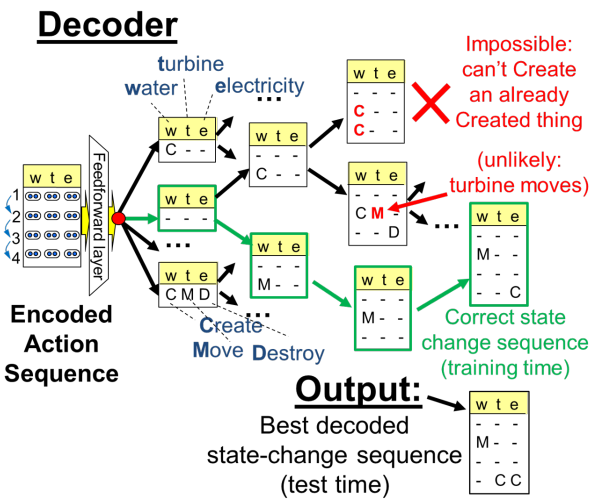

Figure 4: The decoder, illustrated for the ProPara domain. Each action embedding c_{tj} is first passed through a feedforward layer to generate a distribution over the (here $K = 4$) possible state changes that could result, for each entity (listed horizontally) at each step (listed vertically downwards). The decoder then explores the space of state-change sequences, using these distributions to guide the search. During end-to-end training, ProStruct follows the correct (green) path, and backpropagates to drive up probabilities along this path. During testing, the system performs a beam search to find the most globally plausible sequence.

sequence so far Π_t:

$$\phi'(\pi_{t+1}) = \sum_{j=1}^{|E|} \Big(\lambda \, logit(\pi_{t+1}^j) + (1-\lambda) \, \log P(\pi_{t+1}^j | e_j, topic) \Big) \quad (6)$$

where λ is a learned parameter controlling the degree of bias.

During search, when making a transition along a path from Π_t to a valid Π_{t+1}, Π_{t+1} is scored by accumulating normalized scores along the path:

$$\phi(\Pi_{t+1}) = \phi(\Pi_t) + \frac{\phi'(\pi_{t+1})}{\sum_{\pi'_{t+1} \in \Pi_{t+1}} \phi'(\pi'_{t+1})} \quad (7)$$

Continuing state transitions in this manner, when we reach the finished state (i.e., last sentence), our objective is to maximize the score of the state changes produced when reading each sentence. During training, we only materialize a valid node when $\Pi_t \in \Pi_t^*$ where Π_t^* is the set of nodes along the gold path.

We use this constrained decoding to predict the state change sequence. For state changes that take additional parameters, e.g., in the ProPara model a `move` is parameterized by the before and after

locations, we also predict those parameter values during decoding. This is done using standard span prediction layers (inspired by BiDAF, Seo et al. (2017a)) on top of the encoded input.

The model is trained to minimize the joint loss of predicting the correct state changes and correct state change parameters for every sentence in the paragraph:

$$\mathcal{L} = -\sum_{t=1}^{T}\Big(\log P(\pi_t) + \sum_{j=1}^{|E|}\sum_{p\in \text{param}(\pi_t^j)} \log P(v_{pjt}|\pi_t^j)\Big) \quad (8)$$

where $\text{param}(\pi_t^j)$ are the parameters of state change π_t^j, and v_{pjt} are the values of those parameters. For example, `MOVE` is parameterized by before/after locations, and the 2nd loss term refers to the predicted values of those locations.

At test time, instead of following the gold state change path, we use beam search. After reading any sentence, we explore the top-k states sorted by the score $\phi'(\pi_t)$ that satisfy hard constraints. This way, we predict a sequence of state changes that have maximum score while being sensible w.r.t. hard constraints.

5 Incorporating Commonsense Knowledge

By formulating procedural text comprehension as a structured prediction task, we can introduce commonsense knowledge as hard and soft constraints into the model, allowing nonsensical and unlikely predictions to be avoided, and allowing the system to recover from early mistakes.

5.1 Hard Constraints

Hard constraints are introduced by defining the (boolean) function over a candidate sequence of state changes:

$$\text{allowable}(\Pi)$$

used in Equation 4.

While this function can be defined in any way, for the ProPara application we use six constraints. The first three below are based on basic "laws of physics" or commonsense (CS) and are universally applicable:

CS-1: An entity must **exist** before it can be **moved** or **destroyed**
CS-2: An entity cannot be **created** if it already **exists**
CS-3: An entity cannot change until it is mentioned in the paragraph

The next three constraints are observed in the training data:

D-1: Maximum number of toggles for an entity between Exists and not Exist $\leq f_{max\_toggles}$
D-2: Max fraction of entities that are changed per sentence $\leq f_{entities\_per\_sentence}$
D-3: Max fraction of sentences in which an entity changes $\leq f_{sentences\_per\_entity}$

The thresholds used in D-1, D-2 and D-3 are hyperparameters that can be tuned on the dev set.

5.2 Soft Constraints

Soft constraints are introduced by defining the prior probabilities used in Equation 6:

$$P(\pi^j|e_j, topic)$$

that entity e_j undergoes state change π^j in a sentence of text about *topic*. These probabilities are used to re-rank the candidate event sequences during decoding (see Equation 6).

While any method can be used to estimate these probabilities, we describe our corpus-based approach here. Although it was designed for ProPara, it generalizes easily to other domains, and is itself a contribution of this work. For a given state change π^j, entity e_j, and topic, we first gather a corpus of Web sentences mentioning that topic (using Bing search APIs), then count the number of times x that the entity is described as undergoing that state change (e.g., that `water` is said to `MOVE`). To determine this frequency, we first convert the sentences into a set of SRL frames (verb + role-argument pairs) using an off-the-shelf SRL labeler. We then use an existing rulebase, derived from VerbNet, that contains rules that map SRL frames to state changes, e.g., `e1/ARG0 "absorbs"/VERB e2/ARG1` $\Longrightarrow$ `e2 MOVES` (Clark et al., 2018). Although the rules and SRL labels are incomplete and noisy, redundancy in the corpus provides some robustness when estimating the frequency x. Finally, the observed frequency x is converted to a likelihood using a logistic transformation:

$$P(\pi^j|e_j, topic) = \frac{1}{1 + exp^{-(x-x_0)}} \quad (9)$$

where, x_0 is a hyperparameter tuned on the dev set.

5.3 Commonsense Constraints for New Domains

The commonsense constraints we have used for ProPara are general, covering the large variety of topics contain in ProPara (e.g., electricity, photosynthesis, earthquakes). However, if one wants to

apply ProStruct to other genres of procedural text (e.g., fictional text, newswire articles), or broaden the state change vocabulary, different commonsense constraints may be needed. Note that our model architecture itself is agnostic to the source and quantity of hard and soft constraints. For example, one might leverage commonsense rules from existing ontologies such as SUMO (Niles and Pease, 2001) or Cyc (Lenat et al., 1985) to identify new hard constraints; and our corpus-based method could be extended to cover new state change types should the state change vocabulary be extended.

6 Evaluation

We evaluate our model using the ProPara dataset, and compare against several strong baselines published with the original dataset (Dalvi et al., 2018).

6.1 Evaluation setup

Given a paragraph and set of entities as input, the task is to answer four templated questions, whose answers are deterministically computed from the state change sequence:

Q1. What are the inputs to the process?
Q2. What are the outputs of the process?
Q3. What conversions occur, when and where?
Q4. What movements occur, when and where?

Inputs are defined as entities that existed at the start of the process, but not at the end. Outputs are entities that did not exist at the start, but did at the end. A conversion is when some entities are destroyed and others created. Finally, a movement is an event where an entity changes location.

For each process, as every question can have multiple answers, we compute a separate F1 score for each question by comparing the gold and predicted answers. For Q1 and Q2, this is straightforward as answers are atomic (i.e., individual names of entities). For Q3, as each answer is a 4-tuple (*convert-from, convert-to, location, sentence-id*), some answers may only be partially correct. To score partial correctness, we pair gold and predicted answers by requiring the *sentence-id* in each to be the same, and then score each pair by the Hamming distance of their tuples. For Q4, each answer is also a 4-tuple (*entity, from-location, to-location, sentence-id*), and the same procedure is applied. The four F1 scores are then macro-averaged. The total number of items to predict in the train/dev/test partitions is 7043/913/1095.

6.2 Baselines

We compare results using the following process comprehension models:

Recurrent Entity Networks (EntNet) (Henaff et al., 2017) are a state-of-the-art model for the bAbI tasks (Weston et al., 2015). The model uses a dynamic memory to maintain a representation of the world state as sentences are read, with a gated update at each step. These states are decoded to answer questions after each sentence is read.

Query Reduction Networks (QRN) (Seo et al., 2017b) perform a gated propagation of their hidden state across each time-step. Given a question, the hidden state is used to modify the query to keep pointing to the answer at each step.

ProLocal (Dalvi et al., 2018) predicts the state changes described in individual sentences, and then uses commonsense rules of inertia to propagate state values forwards and backwards in time.

ProGlobal (Dalvi et al., 2018) predicts states of an entity across all time steps. It considers the entire paragraph while predicting states for an entity, and learns to predict location spans at time-step $t + 1$ based on location span predictions at t.

7 Results

7.1 Comparison with Baselines

We compare our model (which make use of world knowledge) with the four baseline systems on the ProPara dataset. All models were trained on the training partition, and the best model picked based on prediction accuracy on the dev partition. Table 1 shows the precision, recall, and F1 for all models on the the test partition. ProStruct significantly outperforms the baselines, suggesting that world knowledge helps ProStruct avoid spurious predictions. This hypothesis is supported by the fact that the ProGlobal model has the highest recall and worst precision, indicating that it is over-generating state change predictions. Conversely, the ProLocal model has the highest precision, but its recall is much lower, likely because it makes predictions for individual sentences, and thus has no access to information in surrounding sentences that may suggest a state change is occurring.

We also examined the role of the constraint rules (both hard and soft) on efficiency. With all rules disabled, the training does not complete even one epoch in more than three hours. Because the number of valid states is exponential in the number of

	Precision	Recall	F1
ProLocal	77.4	22.9	35.3
QRN	55.5	31.3	40.0
EntNet	50.2	33.5	40.2
ProGlobal	46.7	52.4	49.4
ProStruct	74.2	42.1	**53.7**

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

	Precision	Recall	F1
ProStruct	70.4	47.8	56.9
- Soft constraints	61.9	47.4	53.7
- Hard constraints†	69.6	47.0	56.1

† Partial ablation, ablated at test only (training without these is computationally infeasible).

Table 2: Ablating world knowledge on the dev set.

entities, the training is particularly slow on paragraphs with many entities. In contrast, with all rules enabled, training takes less than 10 minutes per epoch. This illustrates that the constraints are not only contributing to the model scores, but also helping make the search efficient.

7.2 Ablations and Analysis

To explore the impact of world knowledge, we also performed two ablations on the dev set: Removing soft constraints (at both training and test time), and a partial ablation of removing hard constraints at test time only - note that hard constraints cannot be removed during training because model training time becomes prohibitively large without them, thus qualifying this second ablation. Table 4 shows that F1 drops when each type of knowledge is removed, illustrating that they are helping. The smaller drop for hard constraints suggests that they have primarily been incorporated into the network during training due to this ablation being partial.

Qualitatively, we compared dev set examples where the predicted event sequence changed, comparing predictions made without world knowledge to those made with world knowledge. For readability, we only show the event type predictions (*M*,*C*,*D*, and *N* (shown as "-")) and not their from-location/to-location arguments. If a prediction changes from *X* (without knowledge) to *Y* (with knowledge), we write this "*X* → *Y*". For cases where the prediction changed, we show incorrect predictions in red, and correct predictions in green.

We first compare predictions made with and without the **BK** (corpus-based background knowledge, the soft constraints). Table 3 shows a paragraph about the process of nuclear-powered electricity generation, in the problematic prediction of the generator moving (*M*) was predicted in the second to last sentence. However, the background knowledge contains no examples of generators being moved. As a result, it drives the probability mass away from the move (*M*) prediction, resulting in a no state change (*N*) prediction instead.

Table 4 shows a second example where, without knowledge, no event was predicted for the spark entity. However, BK contains many examples of sparks being created (reflecting text about this topic), shifting the probability mass towards this prediction, resulting in the correct C (create).

Finally, Table 5 shows an example of a hard constraint preventing a nonsensical prediction (namely, electricity is created after it already exists).

7.3 Error Analysis

There are also many cases where incorrect predictions are made. The main causes are summarized below, and offer opportunities for future work.

Implicit reference is a challenge for ProStruct, where an entity affected by an event is not mentioned until a later sentence in the paragraph. For example, in the following ProPara paragraph snippet about combustion engines:

> "...(3) A spark ignites fuel...(4) The pressure pushes the piston down...."

both `spark` and `pressure` are created in sentence 3, even though `pressure` is not mentioned until the subsequent sentence. Recognizing this type of implicit mention is very hard. It is possible that BK could help in such situations, particularly if `ignite` were often associated with creating pressure in the context of a combustion engines, but we did not see such examples in practice.

A second challenge is **coreference**, in particular when different entities have similar names. For example, again for combustion, a snippet looks:

> ...(2) the fuel is injected... (6) the spent

	Without vs. with BK				
	Fuel	Heat	Turbine	Generator	Elec.
Fuel produces heat.	D	C	-	-	-
... Steam spins turbine.	-	-	-	-	-
Generator is turned.	-	-	-	M → N	-
Makes electricity.	-	-	-	-	C

Table 3: BK improves precision. In a nuclear powered electricity generation scenario, BK drives the probability mass away from the generator movement, as a generator does not generally change location.

	Without vs. with BK		
	Fuel	Air	Spark
Fuels burns in the chamber.	D	-	-
The burning fuel creates energy.	-	-	-
The upward motion cause air ...	-	M	-
The piston compresses the air.	-	-	-
A spark ignites the fuel and air ...	-	-	N → C
...	...	...	...

Table 4: BK improves coverage. BK has a strong signal that a spark is usually created in combustion engines, shifting the probability mass towards spark-creation.

	Without and with constraints		
	Electricity	Signals	...
Electricity enters supply unit.	M	-	-
The supply gives electricity to transistors.	C → D	-	
...	...	...	...
The energy is used to complete ...	-	-	

Table 5: Hard constraints avoid nonsensical predictions. In this example without ***CS*-2**, the electricity is predicted to be created after it already exists (impossible). This mistake is avoided using the constraints.

> fuel is ejected. (7) new fuel is injected....

Here `fuel` and `spent fuel` are the same entity, while `new fuel` is a different entity. Correctly tracking these references is challenging (in this case, ProStruct misidentifies (7) as describing an event on the original `fuel/spent fuel`).

A third, related problem is **pronoun resolution**. For example, in:

> The sound continues to bounce off of things and produce echoes until it is totally absorbed or dissipated.

the word `it` confuses ProStruct, and it predicts that the `echo` (rather than the `sound`) is destroyed. We observe several such failure cases.

Finally, we observed **BK retrieval failures** when there was appropriate background knowledge that was expressed in a lexically different way. Consider the example in Table 6 about oil formation. Without BK, the model correctly predicts that `sediment` is destroyed (D). However, BK has few examples of `sediment` being destroyed, and so biases the prediction away from this (correct) choice to an incorrect choice. Further examination of BK shows that it does in fact have knowledge about this destruction, but that is expressed using the word `deposit` instead (e.g., "deposits break down"). A soft (neural) means of accessing BK would help alleviate this problem.

8 Conclusions

Answering questions about procedural text remains challenging, requiring models of actions and the state changes they produce. Predictions made locally throughout the text may together be globally inconsistent or improbable. We have shown how the predicted effects of actions can be improved by treating the task as a structured prediction problem, allowing commonsense knowledge to be injected to avoid an overall inconsistent or improbable set of predictions. In particular, we have shown how two kinds of knowledge can be exploited: hard constraints to exclude impossible and nonsensical state changes, and soft constraints to encourage likely state changes. The resulting system significantly outperforms previous state-of-the-art systems on a challenging dataset, and our ablations and analysis suggest that the knowledge is playing an important role. Our code is available at `https://github.com/allenai/propara`.

	Without BK vs. with BK		
	Algae	Plankton	Sediment
Algae and plankton die.	D	D	-
The dead algae and plankton ...	-	-	-
The sediment breaks down.	-	-	D → M

Table 6: BK lookup limitation: though BK knows that `deposits` can be destroyed (broken down), it does not equate this with (synonymous) `sediments` being destroyed, hence biases model away from correct answer.

Acknowledgements

We thank Oren Etzioni for his insightful feedback and encouragement for this work. We are grateful to Paul Allen whose long-term vision continues to inspire our scientific endeavors.

References

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2014. Neural machine translation by jointly learning to align and translate. *CoRR*, abs/1409.0473.

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)*.

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*.

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 *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 1595–1604, New Orleans, Louisiana. Association for Computational Linguistics.

Mikael Henaff, Jason Weston, Arthur Szlam, Antoine Bordes, and Yann LeCun. 2017. Tracking the world state with recurrent entity networks. In *ICLR*.

Chris Hokamp and Qun Liu. 2017. Lexically constrained decoding for sequence generation using grid beam search. In *ACL*.

Chloé Kiddon, Ganesa Thandavam Ponnuraj, Luke Zettlemoyer, and Yejin Choi. 2015. Mise en place: Unsupervised interpretation of instructional recipes. In *Proc. EMNLP'15*, pages 982–992.

Chloé Kiddon, Luke Zettlemoyer, and Yejin Choi. 2016. Globally coherent text generation with neural checklist models. In *Proc. EMNLP'16*, pages 329–339.

Jayant Krishnamurthy, Pradeep Dasigi, and Matthew Gardner. 2017. Neural semantic parsing with type constraints for semi-structured tables. In *EMNLP*.

Douglas B Lenat, Mayank Prakash, and Mary Shepherd. 1985. Cyc: Using common sense knowledge to overcome brittleness and knowledge acquisition bottlenecks. *AI magazine*, 6(4):65.

Fei Liu and Julien Perez. 2017. Dialog state tracking, a machine reading approach using memory network. In *EACL*.

Ian Niles and Adam Pease. 2001. Towards a standard upper ontology. In *FOIS*.

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.

Hannah Rashkin, Antoine Bosselut, Maarten Sap, Kevin Knight, and Yejin Choi. 2018. Modeling naive psychology of characters in simple commonsense stories. 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*.

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*.

Spacy. 2018. Spacy tokenizer API reference page. https://spacy.io/api/annotation#pos-tagging. Accessed: 2018-04-10.

Zhaopeng Tu, Zhengdong Lu, Yang Liu, Xiaohua Liu, and Hang Li. 2016. Coverage-based neural machine translation. In *ACL*.

Oriol Vinyals, Lukasz Kaiser, Terry Koo, Slav Petrov, Ilya Sutskever, and Geoffrey E. Hinton. 2015. Grammar as a foreign language. In *NIPS*.

Tsung-Hsien Wen, Milica Gasic, Nikola Mrksic, Pei hao Su, David Vandyke, and Steve J. Young. 2015. Semantically conditioned lstm-based natural language generation for spoken dialogue systems. In *EMNLP*.

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*.

Sam Wiseman and Alexander M. Rush. 2016. Sequence-to-sequence learning as beam-search optimization. In *EMNLP*.

Pengcheng Yin and Graham Neubig. 2017. A syntactic neural model for general-purpose code generation. In *ACL*.