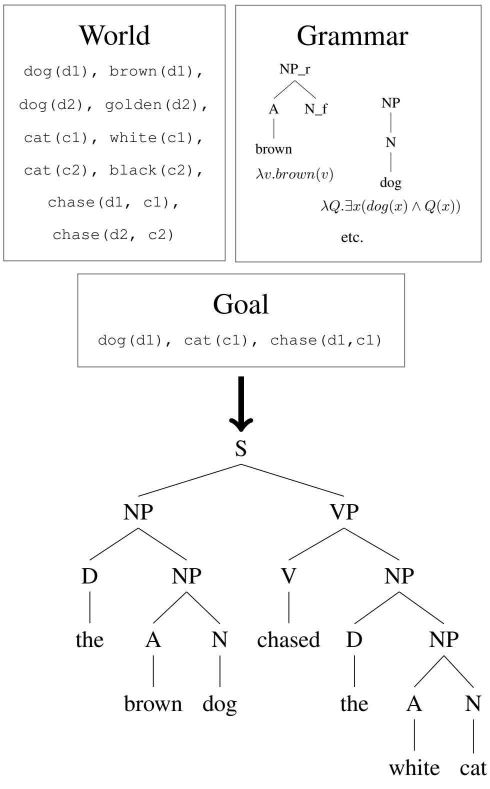
Hybrid Semantics for Goal-Directed Natural Language Generation

Connor Baumler
Case Western Reserve University
baumler@umd.edu

Soumya Ray
Case Western Reserve University
sray@case.edu

Abstract

We consider the problem of generating natural language given a communicative goal and a world description. We ask the question: is it possible to combine complementary meaning representations to scale a goal-directed NLG system without losing expressiveness? In particular, we consider using two meaning representations, one based on logical semantics and the other based on distributional semantics. We build upon an existing goal-directed generation system, S-STRUCT, which models sentence generation as planning in a Markov decision process. We develop a hybrid approach, which uses distributional semantics to quickly and imprecisely add the main elements of the sentence and then uses first-order logic based semantics to more slowly add the precise details. We find that our hybrid method allows S-STRUCT's generation to scale significantly better in early phases of generation and that the hybrid can often generate sentences with the same quality as S-STRUCT in substantially less time. However, we also observe and give insight into cases where the imprecision in distributional semantics leads to generation that is not as good as using pure logical semantics.



$$\exists z1, z2(dog(z1) \wedge brown(z1) \wedge cat(z2)$$
$$\wedge white(z2) \wedge chase(z1, z2))\ [z1 : \mathtt{d1}, z2 : \mathtt{c1}]$$

Figure 1: Example of the goal-directed NLG task. Given a world of facts, a grammar, and a communicative goal, generate a grammatical sentence that unambiguously expresses the goal.

1 Introduction

We consider the problem of goal-directed natural language generation (NLG) (Gatt and Krahmer, 2018). Here, the agent intends to communicate some information about its world to another entity. It has semantic representations for its world, its goal, and a grammar to realize the language. Given this input, the goal is to generate (realize) a syntactically correct representation of the semantic goal without omissions or additions (see Figure 1). This task is different from open ended text generation that fills in text after a prompt or the problem of filling in a blank given some context.

Many previous systems for goal-directed NLG use first-order logic (FOL) extended with the λ-calculus to represent semantics (Church, 1985). This semantic representation allows for very precise generation. However, the process is usually slow, primarily because each step of the generation process needs to check that the semantics of the partially realized text is compatible with the eventual goal. This step typically involves checking all possible compatible *bindings*, which is combinatorial. Using distributional semantic representations (Deerwester et al., 1990; Mikolov et al., 2013; Pennington et al., 2014) may allow us to sidestep this combinatorial process through checks via sim-

Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics
Volume 1: Long Papers, pages 1936 - 1946
May 22-27, 2022 ©2022 Association for Computational Linguistics

ple algebraic operations. However, these semantics may lack precision and introduce errors in generation with respect to the goal. In this paper we ask whether it is possible to combine these two different semantic representations in a single generation system that takes advantage of their strengths while mitigating their weaknesses. In particular our insight is that, early in the generating process, we may not need to be very precise. We can use distributional semantics to quickly add in the main elements of a sentence and then use logical semantics to fill in the details more slowly and precisely. Our goal is to balance these elements to get a more scalable generation system while not sacrificing much, if any, expressiveness.

A rich literature exists for generation systems. Overgeneration and ranking systems derive possible sentences from word lattices (Langkilde-Geary, 2002; Langkilde, 2000; Bangalore and Rambow, 2000). These word lattices are directed acyclic graphs whose edges correspond to single words. To generate a valid sentence, the system can traverse a path in the lattice. Then, they rank the candidate sentences using a language model. An alternative approach is to view generation as an AI planning problem. The planner can apply grammar "actions" to take planning steps until it finds a state that fulfils some communicative goal. One such system is SPUD (Sentence Planner Using Descriptions) which answers questions using a knowledge base (Stone and Doran, 1997). The CRISP system builds on SPUD by applying an off-the-shelf planner instead of using a greedy search (Koller and Stone, 2007; Koller and Hoffmann, 2021). This allows for the application of search heuristics and other advances in classical planning. A further improvement is PCRISP which allows probabilistic actions by translating probabilities into costs (Bauer and Koller, 2010).

Other work uses neural networks with an encoder and/or decoder architectures. For instance, a transformer will already have the semantics of individual words as static word vectors (Vaswani et al., 2017; Peters et al., 2018; Radford et al., 2018; Devlin et al., 2019; Lewis et al., 2020). The overall meaning is calculated using attention and feed-forward layers. These approaches create a complex representation of the language model in the encoder and employ a variety of sampling strategies in the decoder. While they can be much faster than logic-based systems, it can be difficult to guarantee that the generated string will be consistent with some world or goal. Recent work has started to explore hybrid approaches in this space. One approach adds logical constraints and plans to transformers and LSTMs. DualEnc models are provided "content plan" traversals through RDF graphs as input (Zhao et al., 2020). While these plans provide the model with the information that is supposed to be included in the generated text, there is no guarantee the model will include all of it or that the information is truly consistent with the original RDF graph.

Rather than providing a plan before generation, NeuroLogic Decoding constrains generated transformer output based on logical constraints during the decoding step (Lu et al., 2021). By leveraging predicate logic, this allows the output to be more precisely constrained to include any necessary true facts and leave out any extra, potentially incorrect, information. This precision is added at a cost asymptotically equivalent to a conventional beam search. However, the logical constraints in this work are syntactic rather than semantic, and ensure that, for example, certain words are not used by the decoded string.

In contrast to such approaches, in our work, we modify S-STRUCT (McKinley and Ray, 2014; Pfeil and Ray, 2016), a planning based system for goal-directed NLG, to use both distributional as well as logical semantics. S-STRUCT is described in detail in the next section. This system models generation as planning in a Markov decision process (MDP). We show experimentally that our approach scales better than pure logical semantics in many cases. We also identify and discuss tradeoffs that arise from the use of distributional semantics, that in some cases lead to worse generation quality.

2 S-STRUCT

Scalable Sentence Tree Realization using UCT (Upper Confidence bounds applied to Trees), or S-STRUCT, is a planning based NLG system that generates single sentences, using a world of facts, a communicative goal, and a grammar. The world and goal are both specified semantically in FOL. The world describes all entities and relations known to the generator while the goal specifies information to communicate. The grammar consists of a semantically annotated probabilistic lexicalized tree adjoining grammar (PLTAG) derived from the XTAG project (XTAG Research Group, 1998).

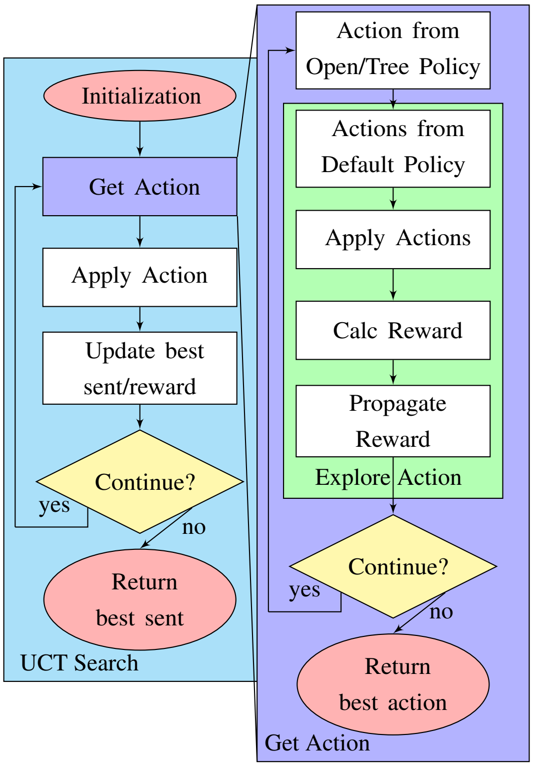


Figure 2: S-STRUCT algorithm. UCT is used to find the best action (PLTAG tree substitution/adjoin) at each step until the goal is satisfied (reward is maximized).

Before S-STRUCT begins the generation process, it finds an expanded communicative goal. The expanded goal includes any extra information necessary to make the goal entities unambigiuous from the other entities in the world. Then, S-STRUCT prunes the grammar of lexicalized trees that, given the goal, will never be used. The resulting pruned grammar will be able to express relations between goal entities and will contain trees that satisfy semantic constraints that may not be explicitly mentioned in the goal (e.g., a complementizer "that" tree). It will also be able to express at least one referring-expression for each unique entity (e.g., if we need to generate a "cat" entity, we may need to clarify whether it is *black* or *brown*).

We observe that once trees are pruned, any associated entities and relations in the world can be pruned as well. This world pruning reduces the space of possible bindings, which we have empirically verified results in a significant speed increase without impacting accuracy. We call this version of S-STRUCT with world pruning S-STRUCT v2.

Once this pruning is completed, to generate a sentence, S-STRUCT uses the UCT (Kocsis and Szepesvári, 2006) procedure (see Figure 2) to plan in an MDP. States in the MDP are semantically annotated partial trees reflecting the partial sentence constructed so far. Actions adjoin or substitute a single PLTAG tree. At each step, S-STRUCT ranks actions to add a new fragment to the current partial tree. If there are unexplored actions, it chooses such an action to explore. Otherwise, it chooses actions based on the UCT ranking, which balances explore/exploit criteria.

To estimate the downstream quality of the action, S-STRUCT looks ahead by a number of exploratory actions that are uniformly sampled. For each action, S-STRUCT finds the reward of each state reached, propagating the rewards up the search tree. These rewards identify the best action at a state. The reward is largely determined by how well the partial sentence matches the semantics of the goal. To do this, S-STRUCT considers bindings between entities in the partial sentences and the goal. Here, a valid binding between entities is one in which the stated semantic information does not disagree (e.g., we have a cat in the partial sentence that is *white* and a cat in the goal that is *white* and *long-haired*). In the reward, S-STRUCT only receives credit when a goal entity has a valid biding to a partial-sentence entity.

The reward also considers the number of entities missing a determiner, the number of partial-sentence entities with no goal bindings, the number of world bindings, and the length of the sentence. The first two characteristics penalize missing information. The number of world bindings reflects potential ambiguity in the sentence. Finally, the last criterion reflects the fact that given two sentences, both of which express the goal semantics precisely, we prefer the shorter sentence.

The action search procedure returns the action with the best reward. S-STRUCT applies this action and updates the partial sentence. If a terminal state is reached in which adding more actions will not improve the reward or the generation process runs out of time, then this sentence is returned. If not, the action search repeats. In subsequent searches, we may be able to reuse parts of the search tree of exploratory actions as some will still be relevant in the new state.

To improve search efficiency, the search in S-STRUCT is carried out in two phases. First, only substitution actions are considered until all substitution nodes in the PLTAG tree are filled. Then, adjoin actions (and some substitutions if required by the added adjoins) are considered to complete the generation. This reduces the branching factor of the search considerably, speeding up the process.

3 HS-STRUCT: S-STRUCT with Hybrid Semantics

In this section, we describe how we modify S-STRUCT to use distributional semantics. We first describe how we compose distributional semantics and how the distributional reward is computed. Then, we describe additional modifications required by using imprecise semantics in the search, including a step to correct word ordering errors and using a beam search instead of a greedy search in UCT.

3.1 Distributional Composition

We must be able to compose embeddings to obtain the semantics of the goal and each state. We first note that we cannot use order-dependent composition of word vectors to obtain state or goal embeddings. Say our goal is to express $dog(x) \wedge cat(y) \wedge rat(z) \wedge chase(y, z) \wedge chase(x, y)$. If we have generated to the point of, for example, "dog chase cat," then we can use an order dependent method to compose the semantics of the *dog*, *chase*, and *cat* vectors to represent the fragment. However, the logical goal does not order relations in a meaningful way. In fact, figuring out the syntactic structure to realize the goal is a problem S-STRUCT itself solves. So, to compose the goal embedding, we will need a method that is not order dependent. To be consistent, we need to apply the same method for state embeddings as well.

In our approach, we find goal or state embeddings by averaging the components. In other words, we map the entities and relations in our goal or state to word embeddings (for example, the relation "chase" is mapped to the vector for the word *chase*) and then average these to create an embedding for the state or the goal.

3.2 Distributional Reward

For each partial state, we need to compute a reward that measures how close we are to realizing the goal. This is described in Algorithm 1. First, we calculate the distance between the partial state and the goal as the Euclidean norm of the difference between the embeddings (line 2). We next add a penalty for the number of missing or extra conditions (line 3) and the sentence length (line 4). Thus the best states will be short sentences that do not have missing or extra conditions and that have embeddings close to the goal. C_1, C_2 and C_3 are weight factors that modify the relative importance of these factors (hand-selected as 100, 15 and 10 and consistent in all experiments). Finally, for S-STRUCT we need the reward to be positive. This is because S-STRUCT's tree policy chooses actions in part based on the total reward over all the times it has been applied. Here, a negative reward will penalize actions that we see more often. To fix this, we add a large constant to the reward (line 1), making the reward always positive. This reward shaping will not affect the optimal plan (Ng et al., 1999).

Algorithm 1: calcReward

```
  Input: Partial Sentence $S$, World $W$, Goal $G$
1 $score \leftarrow R_{max}$
2 $score -= C_1 \| S.Sem - G.Sem \|$
3 $score -= C_2 | G.conds - S.conds |$
4 $score -= C_3 | S.sentence |$
5 return $score$
```

3.3 Integrating Distributional and Logical Semantics

Each semantic representation has its strengths and weaknesses. How should we integrate the two? First, consider an alternative in which we only use distributional semantics. This version (let us call it "PureDist") would be fast, but runs into several issues. First, in the absence of word-order-sensitive composition, PureDist cannot identify which sentences have the wrong word ordering.

Additionally, a problem arises with the stopping criterion. Generation should stop when not taking an action leads to a better reward than taking one. For PureDist, since our embedding vectors are high dimensional, there are many degrees of freedom to slightly improve the reward. So, while S-STRUCT can only get reward for fulfilling a goal relation or adding a determiner to an entity once, PureDist can keep generating by adding new, potentially repetitive words that are not adding any new information but instead are moving the state slightly closer to the goal vector. Without a strong way to determine whether or not the current state has reached the goal, PureDist's generation quality is more heavily tied to the balance of the sentence length penalty.

If the sentence length penalty is too high, PureDist will cut generation off before useful information has been expressed. If it is too low, generation can continue to add irrelevant words that move the state slightly closer to the goal. As a result, using a purely distributional semantic search is not a viable alternative (this is validated in our experiments).

This leaves two options: we can use distributional semantics to start, and then switch to logical, or vice versa. Of these, the first is more suitable. The key intuition is that early in generation, the work done to compute bindings and to validate partial sentences in S-STRUCT is overkill. A less precise semantic representation could do just as well, while being more efficient. Further, by switching to logical semantics *after* distributional we will have the opportunity to correct word ordering errors. Conversely, using logical semantics in the early phase means we are potentially doing unnecessary work. Therefore we decide to use distributional semantics to start, and then switch to logical semantics.

When should we switch? As mentioned above, S-STRUCT has a natural transition point. The first part of the search focuses only on substitution actions (a "substitution phase") before switching to an "adjoin phase." We choose to use distributional semantics in the substitution phase. In our example in Figure 3, we use our distributional reward in the initial and substitution actions getting us to "cat chased dog." At this point, we cannot add more information without adjoin actions, so we can move to the next phase.

3.4 Swap Actions

Since our state/goal composition is word order independent, the output of the first phase may have ordering errors, such as in Figure 3b. We address this by adding a Swap phase in between the distributional and logical semantic phases. In this phase, we find all pairs of entities in the tree output by the distributional phase that have the same type (such as a noun phrase), and consider the trees that result if we exchange them. For each such tree, we compute the original S-STRUCT reward with logical semantics. This means that we consider exact bindings of the sentence entities to their world and goal counterparts to determine the reward of the sentence. We greedily apply the best swaps we can find until doing no swap yields a better reward. Unlike with Dist, Swap will only get credit for adding an entity if it is being used correctly, meaning Swap will get a better reward when word ordering mistakes are fixed.

3.5 Beam Search

While Swap actions can mitigate some of the mistakes caused by the first phase, they may not account for all possible errors, such as the use of incorrect substitutions. So we use a beam search within the UCT search of the first phase of HS-STRUCT instead of greedily selecting the best state. This means HS-STRUCT can keep track of multiple states that may seem sub-optimal when using the distributional reward but will be more successful under the formal logic reward. The resulting beam after the first phase is passed into Swap as described above. Each partial state is processed by Swap, and the best state found is then input to the third phase, which is regular S-STRUCT, to perform adjoins and finish the generation process (shown in Figure 3c and 3d). By adding this beam search, we allow HS-STRUCT to partially underspecify substitution decisions during the distributional phase.

To keep generation efficient, we split trials of exploratory actions between all states in the beam. In other words, each beam search state uses an equal portion of the overall exploratory actions, keeping the total number of exploratory actions the same as without the beam search.

3.6 HS-STRUCT Algorithm

Our HS-STRUCT algorithm is shown in Algorithm 2. We begin by using distributional semantics (Dist) from the initial (empty) state. This search follows the general structure of the original S-STRUCT search (Figure 2) though with a beam search (line 4 and Figure 3a). We only allow initial and substitution actions in this phase as we are only trying to block out the main ideas. Then, we consider swap actions to correct for word order issues (lines 5 and 6 and Figure 3b). Now that we have our swapped states, we down select into a single state for the remainder of generation (line 7). Finally, we use our original FOL-based S-STRUCT to add any details that would have required adjoin actions to finish out our generation (line 8 and Figures 3c and 3d).

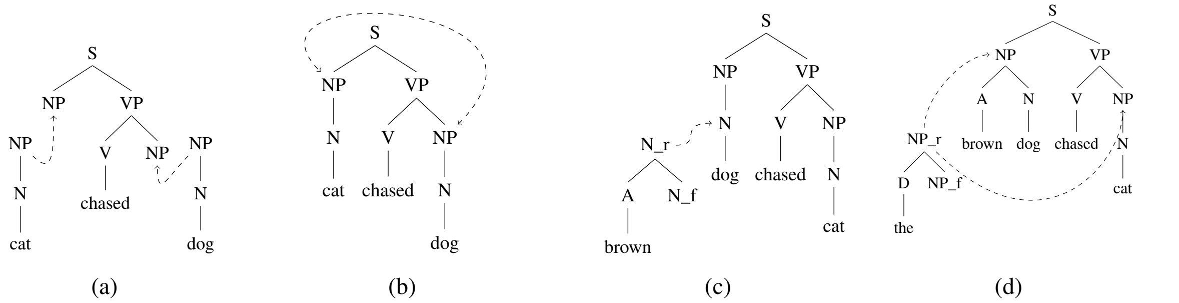


Figure 3: Example of HS-STRUCT generation of "The brown dog chased the cat." (a) Using the distributional reward, we do the initial "chased" action and substitute the "dog" and "cat". (b) We use a swap action to fix the word ordering mistake. (c&d) Using the logical reward, we adjoin the "brown" detail and the determiners.

```
Algorithm 2: HS-STRUCT
  Input: Grammar $R$, World $W$, Goal $G$,
         NumTrials $N$, Lookahead $D$,
         Timeout $T$, Beam Width $B$
  $R \leftarrow pruneGrammar(R)$
  $W \leftarrow pruneWorld(W, R)$
  $states \leftarrow$ a size $B$ empty list
  $states \leftarrow Dist(R, W, G, N, D, T, states)$
  for i in [1,B] do
      $states[i] \leftarrow Swap(states[i])$
  $state \leftarrow$
    $\arg\max_{s \in states} calcLogicalReward(s)$
  $bestSentence, state \leftarrow$
    $FOL(R, W, G, N, D, T, state)$
  return $bestSentence$
```

4 Empirical Evaluation

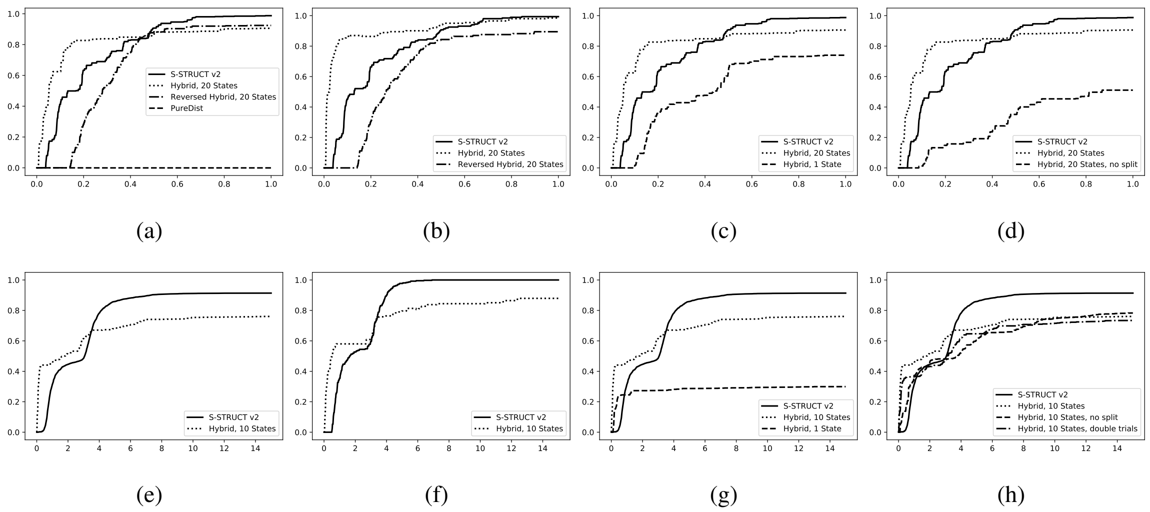
Our primary hypothesis is that integrating distributional and logical semantics through HS-STRUCT will scale better (i.e. generate better quality sentences in less time) than either S-STRUCT v2 (S-STRUCT with world pruning), PureDist or using distributional semantics after logical semantics. We will also evaluate the impacts of design choices such as the beam search. We have not compared against contextual language models in our experiments because, as described in Section 1, the most related such approaches that we know of still do not address the goal-directed NLG task.

Data. We follow prior work and focus on generation of English sentences, pulling world facts and goals from the WSJ section of the Penn Tree-Bank corpus (McKinley and Ray, 2014; Marcus et al., 1999). The sentences were parsed with an LTAG parser (Sarkar, 2000; XTAG Research Group, 1998) to find the best parse trees for each sentence. A subset of the most frequently occurring XTAG trees were chosen and manually semantically annotated with FOL semantics. Together, these trees could parse 74% of the corpus. For our distributional semantic representation, we use pre-trained OLIVE word vectors trained on the Wikipedia English corpus (Seonwoo et al., 2019). While HS-STRUCT is agnostic to the distributional semantic representation used, OLIVE vectors are trained to have additive compositionality, so we know we can compose our partial sentence and goal embeddings. Some sentences in our dataset had to be removed because of the lack of OLIVE vectors to cover them.

Test Set	Goals	World Entities	World Relations	Avg. Goal Entities	Avg. Goal Relations
Simple	32	48	36	1.50	1.13
Complex	750	1224	1245	1.63	1.66

Table 1: Summary statistics for evaluation datasets

For our experiments, we choose goals from the semantic annotations of sentences in the dataset, with the world being a combination of facts from the semantics of all goals (sentences). These worlds and goals are split into a "simple" and a "complex" dataset based on the complexity of the goals with the complex dataset having more world entities, relations, average relations and entities in the goal (Table 1). These datasets are made from non-overlapping goals. An example simple goal is $bank(z1) \wedge acquiesced(z1)$ which could mean "The bank acquiesced.", and an example complex goal is $siliconvalley(z1) \wedge sigh(z2) \wedge relief(z3) \wedge heaved(z1, z2) \wedge of(z2, z3)$ which could mean "Silicon Valley heaved a sigh of relief."



(a) (b) (c) (d)

(e) (f) (g) (h)

Figure 4: Comparison of reward gained or ROUGE-1 score and time to generate using various versions of S-STRUCT. (a-d) and (e-f) use the simple and complex datasets respectively. The x-axes show time in seconds and the y-axes show the percentage of the best metric at a given time. In all cases, scores are averaged over all goals in the given dataset. (a) and (e) show the overall reward gain on HS-STRUCT vs other versions of S-STRUCT. (b) and (f) similarly show the changes in ROUGE-1 score. (c) and (g) look at the effect of using the beam search. (d) and (h) show the effect of splitting or not splitting trials between stored states of HS-STRUCT.

Metrics. There are two possible ways to evaluate our generation quality: syntactic and semantic. The reward assigned by S-STRUCT v2 and HS-STRUCT is primarily based on a semantic match between the goal and the partial sentence. However, in some cases a semantic difference can be misleading if the syntactic realization is similar. So, in addition to our semantic reward, we also evaluate our sentences using ROUGE-1 (Recall-Oriented Understudy for Gisting Evaluation)(Lin, 2004). ROUGE is designed to evaluate summaries of texts by comparing them to ideal summaries created by humans. We use it to compare the result of each approach after each action to the "ideal" sentence with the best possible reward.

All experiments were implemented in Python 3.7.1. They were run on a single core of an Intel(R) i5-8250 processor clocked at 1.60GHz with access to 8GB of RAM. The results of our evaluation are shown in Figure 4a to 4h. In each case the x-axis is time in seconds and the y-axis is the percentage of the best metric at a given time. The results are averaged over all goals in each dataset.

4.1 HS-STRUCT vs. PureDist

As we can see in Figure 4a, HS-STRUCT gains a much higher reward than the method that only uses the distributional reward, PureDist. As we discussed in Section 3.3, there are a number of issues like word ordering and stopping criteria which makes generation with only distributional semantics difficult. This result provides empirical validation of these observations. Because PureDist performed so poorly on even simple goals, we did not run it on complex goals.

4.2 HS-STRUCT vs. Reversed Hybrid

We also consider reversing the order of distributional and formal semantics within the hybrid, starting with the logic-based FOL and then using the distributional Dist. As we can see in Figure 4a, reversed hybrid is less efficient than either HS-STRUCT or S-STRUCT v2. Reversed HS-STRUCT needs to spend twice the time switching between semantic systems and potentially much more time completing swap actions than regular HS-STRUCT. While we chose the sentence length tradeoff hyperparameter to allow for the reversed HS-STRUCT to achieve a good reward, in general using a distributional reward for adjoin actions leads to issues with deciding when to cut generation off. Overall, this shows that simply reversing HS-STRUCT does not yield an improvement in performance.

4.3 HS-STRUCT vs. S-STRUCT v2

On the simple dataset (Figures 4a and 4b), HS-STRUCT initially creates sentences with a higher syntactic and semantic quality faster than S-STRUCT v2. The final syntactic quality (4b) is the same as S-STRUCT v2, though there is a small gap in the final semantic quality. On the complex dataset, (Figures 4e and 4f), while HS-STRUCT again produces sentences with a higher reward faster than S-STRUCT v2 early on in generation. But there is a decrease in the final metric obtained by HS-STRUCT both syntactically and semantically by about 12% syntactically and about 17% semantically. The reason for the gap in the final metric values is analyzed further below. Overall, we find that HS-STRUCT sometimes produces lower quality sentences on high complexity goals than S-STRUCT v2 given enough time. However, even when the goal is complex, HS-STRUCT can produce higher quality sentences than S-STRUCT v2 under a short time limit, and consistently achieves this when the goal is simple.

4.4 Effect of Beam Search

Figures 4c and 4g show the effect of using a beam search in the distributional phase of HS-STRUCT as opposed to a greedy search. On the simple dataset (Figure 4c), allowing HS-STRUCT to choose between stored states in the formal logic phase improved the average final reward by 22%. On the complex dataset, this change resulted in a 154% increase. This indicates as well that the distributional phase may not be very accurate at selecting the single best state in the search process.

When a beam search is used, a decision must be made how to allocate the UCT rollouts to different states in the beam. Instead of giving each state the same number of trials as the single, greedy search approach, which would make the beam search much less efficient, we hypothesize that we can split the overall number of trials between each state to receive a comparable reward in significantly less time. As we can see in Figure 4d, splitting the number of trials has a very significant effect on the speed and quality of generation on the simple dataset. Simply providing each state in the beam the same number of trials as greedy search slows generation considerably. Again in Figure 4h, we observe that not splitting trials slows generation substantially in the early phases while having no significant impact on quality. Increasing the number of trials to 2x also slowed generation with no significant increase in quality. Thus, a broad and shallow search seems well suited to the early phase of generation, which agrees with our intuition.

4.5 Discussion

Our results show that HS-STRUCT can produce sentences that are around 12% lower in terms of syntactic quality than S-STRUCT v2 under no time constraints. In this section we discuss why we see this gap and whether it is due to fundamental aspects of distributional semantics.

The reward gap between HS-STRUCT and S-STRUCT v2 results from a number of issues such as incorrect parts of speech, incorrect verb valence, and over-generation. These errors can co-occur, but we report them without overlap, prioritizing part of speech errors. This means that the reported error frequencies are a lower bound for every error type except part of speech.

Parts of Speech. The same word may be used as different parts of speech. This is important in generation, but distributional semantics has difficulty telling this apart. A common problem for HS-STRUCT was using an entity as a verb. These mistakes will not be fixed by the Swap phase and will not allow for valid bindings in the logical phase. This means that we cannot recover if all beam search states contain this error. S-STRUCT v2 does not make such mistakes, since it does not use distributional semantics.

On the complex dataset, these part of speech mistakes account for about 50% of cases in which HS-STRUCT earns a worse reward than S-STRUCT v2. Such errors could potentially be fixed with improved embeddings considering different vectors for different parts of speech.

Verb Valence. Our grammar has multiple possible verb trees which will be lexicalized with the possible verbs in our lexicon. This means that, as appropriate, verbs can lexicalize multiple trees, representing the different number of arguments the verb could take (also known as the verb's valence). In S-STRUCT, this valence distinction will not cause issues. Using the wrong tree with the correct root will not count toward the goal satisfaction portion of the reward since the number of arguments in the semantic representation has to match. For HS-STRUCT, however, the reward calculation does not explicitly check for the correct valence as the verb is the same, with the same embedding, so the same goal distance will be given to partial sentences using either tree.

In the simple dataset, this leads to a partially artificial reward gap. In 60% of cases in which HS-STRUCT received a lower reward, the final sentences produced by HS-STRUCT and S-STRUCT v2 are identical. In the complex dataset, valence issues accounted for about 20% of cases in which S-STRUCT v2 outperformed HS-STRUCT. This issue could be alleviated by computing different embeddings for different valences of a verb.

Over-generation. On the complex dataset, we also see a number of cases in which the overall content of the two generated sentences were nearly identical, with HS-STRUCT adding in unneeded additions like extra complementizers. Since extra complementizers do not change the semantic content, this mistake will barely affect the logical reward but does decrease ROUGE-1 scores. This over-generation may also stem from the "incremental benefit" of HS-STRUCT's reward function as described in Section 3.3, and accounts for about 15% of cases in which HS-STRUCT earns a worse reward than S-STRUCT v2 on the complex dataset. It could potentially be alleviated by more carefully tuning the sentence length penalty in the reward.

Examples. Consider the sentence "The rates in the secondary market are typical," which is expressed as the goal $rates(z1) \wedge typical(z1) \wedge market(z2) \wedge secondary(z2) \wedge in(z1, z2)$. As we can see, there is no verb listed in the goal semantics. The copula "are" would not make sense as an FOL relation as $typical(z1)$ already implies that the rates *are* typical. We also run into issues with the preposition "in". Since it could also feasibly be an abbreviation for "inch", our grammar includes it as a noun as well.

HS-STRUCT begins by choosing a "typical" declarative adjective small clause tree (αnx0Ax1) with "typical" as the AP and the NP and V substitution nodes left open. It cannot tell the difference between noun and preposition "in", so it sees substituting "in" for the remaining NP node (essentially creating the string "in␣typical"). Using our OLIVE vectors, the combination of "in" and "typical" is closer to the goal than the correct "rates" and "typical" (because "in" is present in the goal), so this part-of-speech mistake is seen as beneficial. HS-STRUCT may also store the "rates" substitution, but this is a function of the beam width. It leaves the copular verb "are" location blank, as this verb does not appear in the goal.

While the "in" substitution helped the distributional reward, it will hurt the logical reward in future since there is no "in" entity in the world or goal to bind to (there is only an "in" relation). Since HS-STRUCT will not be able to find valid bindings, it will not be able to add additional information, forcing the generation to stop at "in␣typical". Here, generation can be improved by increasing the beam states until the "rates" substitution is also chosen, or by having the distributional phase represent preposition "in" and inches "in" separately.

Another such error is shown by the sentence "Investors dumped any technology shares." Here, the "shares" are an entity in the goal (i.e., written as $instance\_of(x, shares)$ not $shares(x, y)$). HS-STRUCT will represent both the verb "shares" and the noun "shares" the same way, so it does not know that it should not consider the initial tree using the verb "shares". In this case, HS-STRUCT ends the distributional phase with the best sentence "Investors share technology." Again, such an error is not recoverable in the logical semantics phase.

A different issue that may contribute to HS-STRUCT's lower reward in some cases is that of copular verbs. These do not appear in the goal semantics. However, if there is some non-copular verb in the goal, HS-STRUCT may incorrectly substitute in a verb in the copular verb slot, as doing so will decrease the goal distance. If the beam search did not keep a state without one of these incorrect substitutions, then HS-STRUCT will not be able to recover in the logical phase.

5 Conclusion

We have presented HS-STRUCT, which uses both distributional and logical semantics for goal-directed language generation. By taking a hybrid approach HS-STRUCT's generation scales significantly better in early phases. However, in some cases, the quality of the final generation can be lower than a pure logical approach. HS-STRUCT is available through GitHub upon request.

References

Srinivas Bangalore and Owen Rambow. 2000. Exploiting a probabilistic hierarchical model for generation. In *COLING 2000 Volume 1: The 18th International Conference on Computational Linguistics*.

Daniel Bauer and Alexander Koller. 2010. Sentence generation as planning with probabilistic LTAG. In *Proceedings of the 10th international workshop on* *tree adjoining grammar and related frameworks (TAG+ 10)*, pages 127–134.

Alonzo Church. 1985. *The Calculi of Lambda-Conversion*, volume 6 of *Annals of Mathematics Studies*. Princeton University Press.

Scott Deerwester, Susan T. Dumais, George W. Furnas, Thomas K. Landauer, and Richard Harshman. 1990. Indexing by latent semantic analysis. *Journal of the American Society for Information Science*, 41(6):391–407.

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.

Albert Gatt and Emiel Krahmer. 2018. Survey of the state of the art in natural language generation: Core tasks, applications and evaluation. *Journal of Artificial Intelligence Research*, 61(1):65–170.

Levente Kocsis and Csaba Szepesvári. 2006. Bandit based monte-carlo planning. In *Machine Learning: ECML 2006*, pages 282–293, Berlin, Heidelberg. Springer Berlin Heidelberg.

Alexander Koller and Joerg Hoffmann. 2021. Waking up a sleeping rabbit: On natural-language sentence generation with FF. *Proceedings of the International Conference on Automated Planning and Scheduling*, 20(1):238–241.

Alexander Koller and Matthew Stone. 2007. Sentence generation as a planning problem. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, pages 336–343, Prague, Czech Republic. Association for Computational Linguistics.

Irene Langkilde. 2000. Forest-based statistical sentence generation. In *1st Meeting of the North American Chapter of the Association for Computational Linguistics*.

Irene Langkilde-Geary. 2002. An empirical verification of coverage and correctness for a general-purpose sentence generator. In *Proceedings of the International Natural Language Generation Conference*, pages 17–24, Harriman, New York, USA. Association for Computational Linguistics.

Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. 2020. BART: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7871–7880, Online. Association for Computational Linguistics.

Chin-Yew Lin. 2004. Rouge: A package for automatic evaluation of summaries. In *Text summarization branches out*, pages 74–81.

Ximing Lu, Peter West, Rowan Zellers, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. 2021. NeuroLogic decoding: (un)supervised neural text generation with predicate logic constraints. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 4288–4299, Online. Association for Computational Linguistics.

Mitchell P Marcus, Beatrice Santorini, Mary Ann Marcinkiewicz, and Ann Taylor. 1999. Treebank-3 LDC99T42.

Nathan McKinley and Soumya Ray. 2014. A decision-theoretic approach to natural language generation. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 552–561, Baltimore, Maryland. Association for Computational Linguistics.

Tomas Mikolov, Wen-tau Yih, and Geoffrey Zweig. 2013. Linguistic regularities in continuous space word representations. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 746–751, Atlanta, Georgia. Association for Computational Linguistics.

Andrew Y. Ng, Daishi Harada, and Stuart J. Russell. 1999. Policy invariance under reward transformations: Theory and application to reward shaping. In *Proceedings of the Sixteenth International Conference on Machine Learning*, page 278–287, San Francisco, CA, USA.

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, Doha, Qatar. Association for Computational Linguistics.

Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations. 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 2227–2237, New Orleans, Louisiana. Association for Computational Linguistics.

Jonathan Pfeil and Soumya Ray. 2016. Scaling a natural language generation system. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1148–1157, Berlin, Germany. Association for Computational Linguistics.

Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. 2018. Improving language understanding with unsupervised learning. Technical report, OpenAI.

Anoop Sarkar. 2000. Practical experiments in parsing using Tree Adjoining Grammars. In *Proceedings of the Fifth International Workshop on Tree Adjoining Grammar and Related Frameworks (TAG+5)*, pages 193–198, Université Paris 7.

Yeon Seonwoo, Sungjoon Park, Dongkwan Kim, and Alice Oh. 2019. Additive compositionality of word vectors. In *Proceedings of the 5th Workshop on Noisy User-generated Text (W-NUT 2019)*, pages 387–396, Hong Kong, China. Association for Computational Linguistics.

Matthew Stone and Christine Doran. 1997. Sentence planning as description using Tree Adjoining Grammar. In *35th Annual Meeting of the Association for Computational Linguistics and 8th Conference of the European Chapter of the Association for Computational Linguistics*, pages 198–205, Madrid, Spain. 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 *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.

XTAG Research Group. 1998. A lexicalized tree adjoining grammar for english. *arXiv preprint cs/9809024*.

Chao Zhao, Marilyn Walker, and Snigdha Chaturvedi. 2020. Bridging the structural gap between encoding and decoding for data-to-text generation. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 2481–2491, Online. Association for Computational Linguistics.