A Decision-Theoretic Approach to Natural Language Generation

Nathan McKinley
Department of EECS
Case Western Reserve University
Cleveland, OH, USA
nath@nmckinley.com

Soumya Ray
Department of EECS
Case Western Reserve University
Cleveland, OH, USA
sray@case.edu

Abstract

We study the problem of generating an English sentence given an underlying probabilistic grammar, a world and a communicative goal. We model the generation problem as a Markov decision process with a suitably defined reward function that reflects the communicative goal. We then use probabilistic planning to solve the MDP and generate a sentence that, with high probability, accomplishes the communicative goal. We show empirically that our approach can generate complex sentences with a speed that generally matches or surpasses the state of the art. Further, we show that our approach is anytime and can handle complex communicative goals, including negated goals.

1 Introduction

Suppose someone wants to tell their friend that they saw a dog chasing a cat. Given such a *communicative goal*, most people can formulate a sentence that satisfies the goal very quickly. Further, they can easily provide multiple similar sentences, differing in details but all satisfying the general communicative goal, with no or very little error. Natural language generation (NLG) develops techniques to extend similar capabilities to automated systems. In this paper, we study the restricted NLG problem: given a grammar, lexicon, world and a communicative goal, output a valid English sentence that satisfies this goal. The problem is restricted because in our work, we do not consider the issue of how to fragment a complex goal into multiple sentences (discourse planning).

Though restricted, this NLG problem is still difficult. A key source of difficulty is the nature of the grammar, which is generally large, probabilistic and ambiguous. Some NLG techniques use sampling strategies (Knight and Hatzivassiloglou, 1995) where a set of sentences is sampled from a data structure created from an underlying grammar and ranked according to how well they meet the communicative goal. Such approaches naturally handle statistical grammars, but do not solve the generation problem in a goal-directed manner. Other approaches view NLG as a *planning* problem (Koller and Stone, 2007). Here, the communicative goal is treated as a predicate to be satisfied, and the grammar and vocabulary are suitably encoded as logical operators. Then automated classical planning techniques are used to derive a plan which is converted into a sentence. This is an elegant formalization of NLG, however, restrictions on what current planning techniques can do limit its applicability. A key limitation is the logical nature of automated planning systems, which do not handle probabilistic grammars, or force ad-hoc approaches for doing so (Bauer and Koller, 2010). A second limitation comes from restrictions on the goal: it may be difficult to ensure that some specific piece of information should *not* be communicated, or to specify preferences over communicative goals, or specify general conditions, like that the sentence should be readable by a sixth grader. A third limitation comes from the search process: without strong heuristics, most planners get bogged down when given communicative goals that require chaining together long sequences of operators (Koller and Petrick, 2011).

In our work, we also view NLG as a planning problem. However, we differ in that our underlying formalism for NLG is a suitably defined *Markov decision process* (MDP). This setting allows us to address the limitations outlined

Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics, pages 552–561,
Baltimore, Maryland, USA, June 23-25 2014.

above: it is naturally probabilistic, and handles probabilistic grammars; we are able to specify complex communicative goals and general criteria through a suitably-defined reward function; and, as we show in our experiments, recent developments in fast planning in large MDPs result in a generation system that can rapidly deal with very specific communicative goals. Further, our system has several other desirable properties: it is an anytime approach; with a probabilistic grammar, it can naturally be used to sample and generate multiple sentences satisfying the communicative goal; and it is robust to large grammar sizes. Finally, the decision-theoretic setting allows for a precise tradeoff between exploration of the grammar and vocabulary to find a better solution and exploitation of the current most promising (partial) solution, instead of a heuristic search through the solution space as performed by standard planning approaches.

Below, we first describe related work, followed by a detailed description of our approach. We then empirically evaluate our approach and a state-of-the-art baseline in several different experimental settings and demonstrate its effectiveness at solving a variety of NLG tasks. Finally, we discuss future extensions and conclude.

2 Related Work

Two broad lines of approaches have been used to attack the general NLG problem. One direction can be thought of as "overgeneration and ranking." Here some (possibly probabilistic) structure is used to generate multiple candidate sentences, which are then ranked according to how well they satisfy the generation criteria. This includes work based on chart generation and parsing (Shieber, 1988; Kay, 1996). These generators assign semantic meaning to each individual token, then use a set of rules to decide if two words can be combined. Any combination which contains a semantic representation equivalent to the input at the conclusion of the algorithm is a valid output from a chart generation system. Other examples of this idea are the HALogen/Nitrogen systems (Langkilde-Geary, 2002). HALogen uses a two-phase architecture where first, a "forest" data structure that compactly summarizes possible expressions is constructed. The structure allows for a more efficient and compact representation compared to lattice structures that were previously used in statistical sentence generation approaches. Using dynamic programming, the highest ranked sentence from this structure is then output. Many other systems using similar ideas exist, e.g. (White and Baldridge, 2003; Lu et al., 2009).

A second line of attack formalizes NLG as an AI planning problem. SPUD (Stone et al., 2003), a system for NLG through microplanning, considers NLG as a problem which requires realizing a deliberative process of goal-directed activity. Many such NLG-as-planning systems use a pipeline architecture, working from their communicative goal through discourse planning and sentence generation. In discourse planning, information to be conveyed is selected and split into sentence-sized chunks. These sentence-sized chunks are then sent to a *sentence generator*, which itself is usually split into two tasks, *sentence planning* and *surface realization* (Koller and Petrick, 2011). The sentence planner takes in a sentence-sized chunk of information to be conveyed and enriches it in some way. This is then used by a *surface realization* module which encodes the enriched semantic representation into natural language. This chain is sometimes referred to as the "NLG Pipeline" (Reiter and Dale, 2000).

Another approach, called *integrated generation*, considers both sentence generation portions of the pipeline together (Koller and Stone, 2007). This is the approach taken in some modern generators like CRISP (Koller and Stone, 2007) and PCRISP (Bauer and Koller, 2010). In these generators, the input semantic requirements and grammar are encoded in PDDL (Fox and Long, 2003), which an off-the-shelf planner such as Graphplan (Blum and Furst, 1997) uses to produce a list of applications of rules in the grammar. These generators generate parses for the sentence at the same time as the sentence, which keeps them from generating realizations that are grammatically incorrect, and keeps them from generating grammatical structures that cannot be realized properly.

In the NLG-as-planning framework, the choice of grammar representation is crucial in treating NLG as a planning problem; the grammar provides the actions that the planner will use to generate a sentence. Tree Adjoining Grammars (TAGs) are a common choice (Koller and Stone, 2007; Bauer and Koller, 2010). TAGs are tree-based grammars consisting of two sets of trees, called initial trees and auxiliary or adjoining trees. An

entire initial tree can replace a leaf node in the sentence tree whose label matches the label of the root of the initial tree in a process called "substitution." Auxiliary trees, on the other hand, encode recursive structures of language. Auxiliary trees have, at a minimum, a root node and a foot node whose labels match. The foot node must be a leaf of the auxiliary tree. These trees are used in a three-step process called "adjoining". The first step finds an adjoining location by searching through our sentence to find any subtree with a root whose label matches the root node of the auxiliary tree. In the second step, the target subtree is removed from the sentence tree, and placed in the auxiliary tree as a direct replacement for the foot node. Finally, the modified auxiliary tree is placed back in the sentence tree in the original target location. We use a variation of TAGs in our work, called a lexicalized TAG (LTAG), where each tree is associated with a lexical item called an anchor.

Though the NLG-as-planning approaches are elegant and appealing, a key drawback is the difficulty of handling probabilistic grammars, which are readily handled by the overgeneration and ranking strategies. Recent approaches such as PCRISP (Bauer and Koller, 2010) attempt to remedy this, but do so in a somewhat ad-hoc way, by transforming the probabilities into costs, because they rely on deterministic planning to actually realize the output. In this work, we directly address this by using a more expressive underlying formalism, a Markov decision process (MDP). We show empirically that this modification has other benefits as well, such as being anytime and an ability to handle complex communicative goals beyond those that deterministic planners can handle.

We note that prior work exists that uses MDPs for NLG (Lemon, 2011). That work differs from ours in several key respects: (i) it considers NLG at a coarse level, for example choosing the type of utterance (in a dialog context) and how to fill in specific slots in a template, (ii) the source of uncertainty is not language-related but comes from things like uncertainty in speech recognition, and (iii) the MDPs are solved using reinforcement learning and not planning, which is impractical in our setting. However, that work does consider NLG in the context of the broader task of dialog management, which we leave for future work.

3 Sentence Tree Realization with UCT

In this section, we describe our approach, called Sentence Tree Realization with UCT (STRUCT). We describe the inputs to STRUCT, followed by the underlying MDP formalism and the probabilistic planning algorithm we use to generate sentences in this MDP.

3.1 Inputs to STRUCT

STRUCT takes three inputs in order to generate a single sentence. These inputs are a grammar (including a lexicon), a communicative goal, and a world specification.

STRUCT uses a first-order logic-based semantic model in its communicative goal and world specification. This model describes named "entities," representing general things in the world. Entities with the same name are considered to be the same entity. These entities are described using first-order logic predicates, where the name of the predicate represents a statement of truth about the given entities. In this semantic model, the communicative goal is a list of these predicates with variables used for the entity names. For instance, a communicative goal of 'red(d), dog(d)' (in English, "say anything about a dog which is red.") would match a sentence with the semantic representation 'red(subj), dog(subj), cat(obj), chased(subj, obj)', like "The red dog chased the cat", for instance.

A grammar contains a set of PTAG trees, divided into two sets (initial and adjoining). These trees are annotated with the entities in them. Entities are defined as any element anchored by precisely one node in the tree which can appear in a statement representing the semantic content of the tree. In addition to this set of trees, the grammar contains a list of words which can be inserted into those trees, turning the PTAG into an PLTAG. We refer to this list as a lexicon. Each word in the lexicon is annotated with its first-order logic semantics with any number of entities present in its subtree as the arguments.

A world specification is simply a list of all statements which are true in the world surrounding our generation. Matching entity names refer to the same entity. We use the closed world assumption, that is, any statement not present in our world is false. Before execution begins, our grammar is pruned to remove entries which cannot possibly be used in generation for the given problem, by tran-

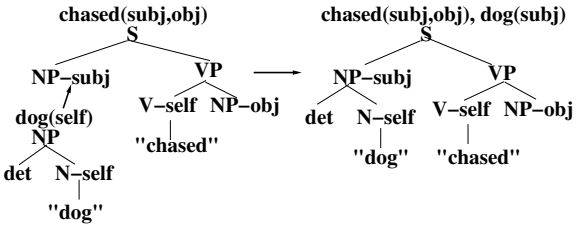


Figure 1: An example tree substitution operation in STRUCT.

sitively discovering all predicates that hold about the entities mentioned in the goal in the world, and eliminating all trees not about any of these. This often allows STRUCT to be resilient to large grammar sizes, as our experiments will show.

3.2 Specification of the MDP

We formulate NLG as a planning problem on a Markov decision process (MDP) (Puterman, 1994). An MDP is a tuple (S, A, T, R, γ) where S is a set of states, A is a set of actions available to an agent, $T : S \times A \times S \rightarrow [0, 1]$ is a possibly stochastic function defining the probability $T(s, a, s')$ with which the environment transitions to s' when the agent does a in state s. $R : S \times A \times S \rightarrow \mathbb{R}$ is a real-valued reward function that specifies the utility of performing action a in state s to reach another state. Finally, γ is a discount factor that allows planning over infinite horizons to converge. In such an MDP, the agent selects actions at each state to optimize the expected infinite-horizon discounted reward.

In the MDP we use for NLG, we must define each element of the tuple in such a way that a plan in the MDP becomes a sentence in a natural language. Our set of states, therefore, will be partial sentences which are in the language defined by our PLTAG input. There are an infinite number of these states, since TAG adjoins can be repeated indefinitely. Nonetheless, given a specific world and communicative goal, only a fraction of this MDP needs to be explored, and, as we show below, a good solution can often be found quickly using a variation of the UCT algorithm (Kocsis and Szepesvari, 2006).

Our set of actions consist of all single substitutions or adjoins at a particular valid location in the tree (example shown in Figure 1). Since we are using PLTAGs in this work, this means every action adds a word to the partial sentence. In situations where the sentence is complete (no nonterminals without children exist), we add a dummy action that the algorithm may choose to stop generation and emit the sentence. Based on these state and action definitions, the transition function takes a mapping between a partial sentence / action pair and the partial sentences which can result from one particular PLTAG adjoin / substitution, and returns the probability of that rule in the grammar.

In order to control the search space, we restrict the structure of the MDP so that while substitutions are available, only those operations are considered when determining the distribution over the next state, without any adjoins. We do this is in order to generate a complete and valid sentence quickly. This allows STRUCT to operate as an anytime algorithm, described further below.

The immediate value of a state, intuitively, describes closeness of an arbitrary partial sentence to our communicative goal. Each partial sentence is annotated with its semantic information, built up using the semantic annotations associated with the PLTAG trees. Thus we use as a reward a measure of the match between the semantic annotation of the partial tree and the communicative goal. That is, the larger the overlap between the predicates, the higher the reward. For an exact reward signal, when checking this overlap, we need to substitute each combination of entities in the goal into predicates in the sentence so we can return a high value if there are any mappings which are both possible (contain no statements which are not present in the grounded world) and mostly fulfill the goal (contain most of the goal predicates). However, this is combinatorial; also, most entities within sentences do not interact (e.g. if we say "the white rabbit jumped on the orange carrot," the whiteness of the rabbit has nothing to do with the carrot), and finally, an approximate reward signal generally works well enough unless we need to emit nested subclauses. Thus as an approximation, we use a reward signal where we simply count how many individual predicates overlap with the goal with *some* entity substitution. In the experiments, we illustrate the difference between the exact and approximate reward signals.

The final component of the MDP is the discount factor. We generally use a discount factor of 1; this is because we are willing to generate lengthy sentences in order to ensure we match our goal. A discount factor of 1 can be problematic in general since it can cause rewards to diverge, but since

there are a finite number of terms in our reward function (determined by the communicative goal and the fact that because of lexicalization we do not loop), this is not a problem for us.

3.3 The Probabilistic Planner

We now describe our approach to solving the MDP above to generate a sentence. Determining the optimal policy at *every* state in an MDP is polynomial in the size of the state-action space (Brafman and Tennenholtz, 2003), which is intractable in our case. But for our application, we do not need to find the optimal policy. Rather we just need to *plan* in an MDP to achieve a *given* communicative goal. Is it possible to do this without exploring the entire state-action space? Recent work answers this question affirmatively. New techniques such as sparse sampling (Kearns et al., 1999) and UCT (Kocsis and Szepesvari, 2006) show how to generate near-optimal plans in large MDPs with a time complexity that is independent of the state space size. Using the UCT approach with a suitably defined MDP (explained above) allows us to naturally handle probabilistic grammars as well as formulate NLG as a planning problem, unifying the distinct lines of attack described in Section 2. Further, the theoretical guarantees of UCT translate into fast generation in many cases, as we demonstrate in our experiments.

Online planning in MDPs as done by UCT follows two steps. From each state encountered, we construct a lookahead tree and use it to estimate the utility of each action in this state. Then, we take the best action, the system transitions to the next state and the procedure is repeated. In order to build a lookahead tree, we use a "rollout policy." This policy has two components: if it encounters a state already in the tree, it follows a "tree policy," discussed further below. If it encounters a new state, the policy reverts to a "default" policy that randomly samples an action. In all cases, any rewards received during the rollout search are backed up. Because this is a Monte Carlo estimate, typically, we run several simultaneous trials, and we keep track of the rewards received by each choice and use this to select the best action at the root.

The tree policy needed by UCT for a state s is the action a in that state which maximizes:

$$P(s,a) = Q(s,a) + c\sqrt{\frac{lnN(s)}{N(s,a)}} \qquad (1)$$

Algorithm 1 STRUCT algorithm.

Require: Number of simulations $numTrials$, Depth of lookahead $maxDepth$, time limit T
Ensure: Generated sentence tree

```
bestSentence ← nil
while time limit not reached do
    state ← empty sentence tree
    while state not terminal do
        for numTrials do
            testState ← state
            currentDepth ← 0
            if testState has unexplored actions then
                Apply one unexplored PLTAG production sampled from the PLTAG distribution to testState
                currentDepth++
            end if
            while currentDepth < maxDepth do
                Apply PLTAG production selected by tree policy (Equation 1) or default policy as required
                currentDepth++
            end while
            calculate reward for testState
            associate reward with first action taken
        end for
        state ← maximum reward testState
        if state score > bestSentence score and state has no nonterminal leaf nodes then
            bestSentence ← state
        end if
    end while
end while
return bestSentence
```

Here $Q(s,a)$ is the estimated value of a as observed in the tree search, computed as a sum over future rewards observed after (s,a). $N(s)$ and $N(s,a)$ are visit counts for the state and state-action pair. Thus the second term is an exploration term that biases the algorithm towards visiting actions that have not been explored enough. c is a constant that trades off exploration and exploitation. This essentially treats each action decision as a bandit problem; previous work shows that this approach can efficiently select near-optimal actions at each state.

We use a modified version of UCT in order to

increase its usability in the MDP we have defined. First, because we receive frequent, reasonably accurate feedback, we favor breadth over depth in the tree search. That is, it is more important in our case to try a variety of actions than to pursue a single action very deep. Second, UCT was originally used in an adversarial environment, and so is biased to select actions leading to the best average reward rather than the action leading to the best overall reward. This is not true for us, however, so we choose the latter action instead.

With the MDP definition above, we use our modified UCT to find a solution sentence (Algorithm 1). After every action is selected and applied, we check to see if we are in a state in which the algorithm could terminate (i.e. the sentence has no nonterminals yet to be expanded). If so, we determine if this is the best possibly-terminal state we have seen so far. If so, we store it, and continue the generation process. Whenever we reach a terminal state, we begin again from the start state of the MDP. Because of the structure restriction above (substitution before adjoin), STRUCT generates a valid sentence quickly. This enables STRUCT to perform as an anytime algorithm, which if interrupted will return the highest-value complete and valid sentence it has found. This also allows partial completion of communicative goals if not all goals can be achieved simultaneously in the time given.

4 Empirical Evaluation

In this section, we compare STRUCT to a state-of-the-art NLG system, CRISP, [1] and evaluate three hypotheses: (i) STRUCT is comparable in speed and generation quality to CRISP as it generates increasingly large referring expressions, (ii) STRUCT is comparable in speed and generation quality to CRISP as the size of the grammar which they use increases, and (iii) STRUCT is capable of communicating complex propositions, including multiple concurrent goals, negated goals, and nested subclauses.

For these experiments, STRUCT was implemented in Python 2.7. We used a 2010 version of CRISP which uses a Java-based GraphPlan implementation. All of our experiments were run on a 4-core AMD Phenom II X4 995 processor clocked at 3.2 GHz. Both systems were given access to 8 GB of RAM. The times reported are from the start of the generation process, eliminating variations due to interpreter startup, input parsing, etc.

[1] We were unfortunately unable to get the PCRISP system to compile, and so we could not evaluate it.

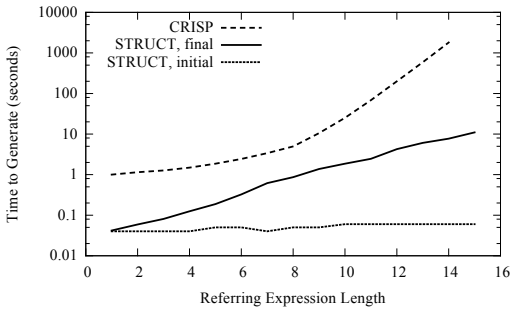


Figure 2: Experimental comparison between STRUCT and CRISP: Generation time vs. length of referring expression

4.1 Comparison to CRISP

We begin by describing experiments comparing STRUCT to CRISP. For these experiments, we use the approximate reward function for STRUCT.

Referring Expressions We first evaluate CRISP and STRUCT on their ability to generate referring expressions. Following prior work (Koller and Petrick, 2011), we consider a series of sentence generation problems which require the planner to generate a sentence like "The Adj_1 Adj_2 ... Adj_k dog chased the cat.", where the string of adjectives is a string that distinguishes one dog (whose identity is specified in the problem description) from all other entities in the world. In this experiment, $maxDepth$ was set equal to 1, since each action taken improved the sentence in a way measurable by our reward function. $numTrials$ was set equal to $k(k+1)$, since this is the number of adjoining sites available in the final step of generation, times the number of potential words to adjoin. This allows us to ensure successful generation in a single loop of the STRUCT algorithm.

The experiment has two parameters: j, the number of adjectives in the grammar, and k, the number of adjectives necessary to distinguish the entity in question from all other entities. We set $j = k$ and show the results in Figure 2. We observe that CRISP was able to achieve sub-second or similar times for all expressions of less than length 5, but its generation times increase exponentially past that point, exceeding 100 seconds for some plans at length 10. At length 15, CRISP failed to generate a referring expression;

after 90 minutes the Java garbage collector terminated the process. STRUCT (the "STRUCT_final" line) performs much better and is able to generate much longer referring expressions without failing. Later experiments had successful referring expression generation of lengths as high as 25. The "STRUCT_initial" curve shows the time taken by STRUCT to come up with the first complete sentence, which partially solves the goal and which (at least) could be output if generation was interrupted and no better alternative was found. As can be seen, this always happens very quickly.

Grammar Size. We next evaluate STRUCT and CRISP's ability to handle larger grammars. This experiment is set up in the same way as the one above, with the exception of l "distracting" words, words which are not useful in the sentence to be generated. l is defined as $j - k$. In these experiments, we vary l between 0 and 50. Figure 3a shows the results of these experiments. We observe that CRISP using GraphPlan, as previously reported in (Koller and Petrick, 2011), handles an increase in number of unused actions very well. Prior work reported a difference on the order of single milliseconds moving from $j = 1$ to $j = 10$. We report similar variations in CRISP runtime as j increases from 10 to 60: runtime increases by approximately 10% over that range.

No Pruning. If we do not prune the grammar (as described in Section 3.1), STRUCT's performance is similar to CRISP using the FF planner (Hoffmann and Nebel, 2001), also profiled in (Koller and Petrick, 2011), which increased from 27 ms to 4.4 seconds over the interval from $j = 1$ to $j = 10$. STRUCT's performance is less sensitive to larger grammars than this, but over the same interval where CRISP increases from 22 seconds of runtime to 27 seconds of runtime, STRUCT increases from 4 seconds to 32 seconds. This is due almost entirely to the required increase in the value of $numTrials$ as the grammar size increases. At the low end, we can use $numTrials = 20$, but at $l = 50$, we must use $numTrials = 160$ in order to ensure perfect generation as soon as possible. Note that, as STRUCT is an anytime algorithm, valid sentences are available very early in the generation process, despite the size of the set of adjoining trees. This time does not change substantially with increases in grammar size. However, the time to perfect this solution does.

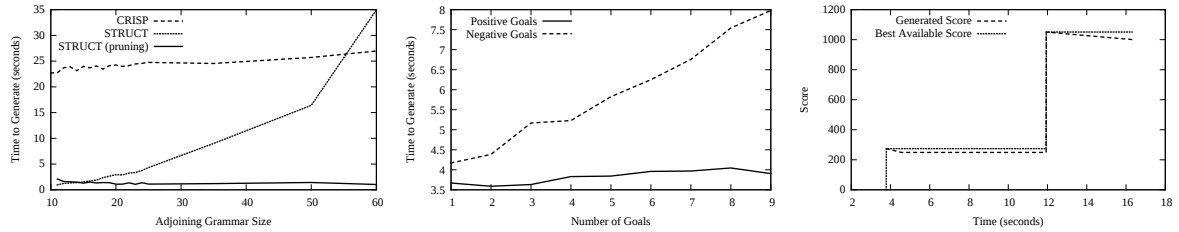
With Pruning. STRUCT's performance improves significantly if we allow for pruning. This experiment involving distracting words is an example of a case where pruning will perform well. When we apply pruning, we find that STRUCT is able to ignore the effect of additional distracting words. Experiments showed roughly constant times for generation for $j = 1$ through $j = 5000$. Our experiments do not show any significant impact on runtime due to the pruning procedure itself, even on large grammars.

4.2 Complex Communicative Goals

In the next set of experiments, we illustrate that STRUCT can solve a variety of complex communicative goals such as negated goals, conjuctions and goals requiring nested subclauses to be output.

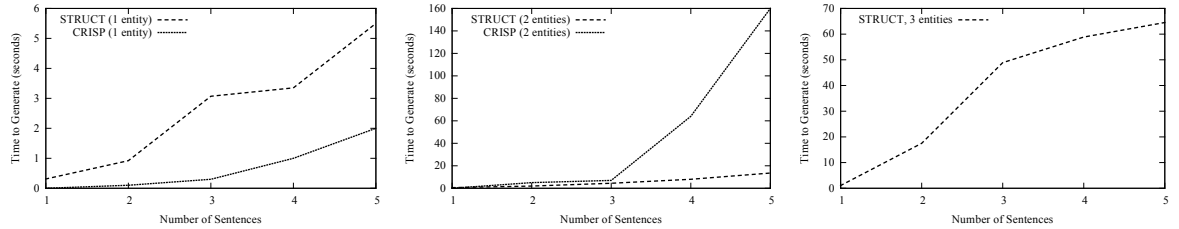
Multiple Goals. We first evaluate STRUCT's ability to accomplish multiple communicative goals when generating a single sentence. In this experiment, we modify the problem from the previous section. In that section, the referred-to dog was unique, and it was therefore possible to produce a referring expression which identified it unambiguously. In this experiment, we remove this condition by creating a situation in which the generator will be forced to ambiguously refer to several dogs. We then add to the world a number of adjectives which are common to each of these possible referents. Since these adjectives do not further disambiguate their subject, our generator should not use them in its output. We then encode these adjectives into communicative goals, so that they will be included in the output of the generator despite not assisting in the accomplishment of disambiguation. For example, assume we had two black cats, and we wanted to say that one of them was sleeping, but we wanted to emphasize that it was a black cat. We would have as our goal both "sleeps(c)" and "black(c)". We want the generator to say "the black cat sleeps", instead of simply "the cat sleeps."

We find that, in all cases, these otherwise useless adjectives are included in the output of our generator, indicating that STRUCT is successfully balancing multiple communicative goals. As we show in figure 3b (the "Positive Goals" curve) , the presence of additional satisfiable semantic goals does not substantially affect the time required for generation. We are able to accomplish this task with the same very high frequency as the CRISP



(a) Effect of grammar size (b) Effect of multiple/ negated goals (c) Effect of nested subclauses

Figure 3: STRUCT experiments (see text for details).



(a) One entity ("The man sat and the girl sat and ..."). (b) Two entities ("The dog chased the cat and ..."). (c) Three entities ("The man gave the girl the book and ...").

Figure 4: Time taken by STRUCT to generate sentences with conjunctions with varying numbers of entities.

comparisons, as we use the same parameters.

Negated Goals. We now evaluate STRUCT's ability to generate sentences given negated communicative goals. We again modify the problem used earlier by adding to our lexicon several new adjectives, each applicable only to the target of our referring expression. Since our target can now be referred to unambiguously using only one adjective, our generator should just select one of these new adjectives (we experimentally confirmed this). We then encode these adjectives into negated communicative goals, so that they will not be included in the output of the generator, despite allowing a much shorter referring expression. For example, assume we have a tall spotted black cat, a tall solid-colored white cat, and a short spotted brown cat, but we wanted to refer to the first one without using the word "black".

We find that these adjectives which should have been selected immediately are omitted from the output, and that the sentence generated is the best possible under the constraints. This demonstrates that STRUCT is balancing these negated communicative goals with its positive goals. Figure 3b (the "Negative Goals" curve) shows the impact of negated goals on the time to generation. Since this experiment alters the grammar size, we see the time to final generation growing linearly with grammar size. The increased time to generate can be traced directly to this increase in grammar size. This is a case where pruning does not help us in reducing the grammar size; we cannot optimistically prune out words that we do not plan to use. Doing so might reduce the ability of STRUCT to produce a sentence which partially fulfills its goals.

Nested subclauses. Next, we evaluate STRUCT's ability to generate sentences with nested subclauses. An example of such a sentence is "The dog which ate the treat chased the cat." This is a difficult sentence to generate for several reasons. The first, and clearest, is that there are words in the sentence which do not help to increase the score assigned to the partial sentence. Notably, we must adjoin the word "which" to "the dog" during the portion of generation where the sentence reads "the dog chased the cat". This decision requires us to do planning deeper than one level in the MDP, which increases the number of simulations STRUCT requires in order to get the correct result. In this case, we require lookahead further into the tree than depth 1. We need to know that using "which" will allow us to further specify which dog is chasing the cat; in order to do this we must use at least $d = 3$. Our reward function must determine this with, at a minimum, the actions corresponding to "which", "ate", and "treat". For these experiments, we use the exact reward function for STRUCT.

Despite this issue, STRUCT is capable of generating these sentences. Figure 3c shows the score of STRUCT's generated output over time for two nested clauses. Notice that, because the exact reward function is being used, the time to generate is longer in this experiment. To the best of our knowledge, CRISP is not able to generate sentences of this form due to an insufficiency in the way it handles TAGs, and consequently we present our results without this baseline.

Conjunctions. Finally, we evaluate STRUCT's ability to generate sentences including conjunctions. We introduce the conjunction "and", which allows for the root nonterminal of a new sentence ('S') to be adjoined to any other sentence. We then provide STRUCT with multiple goals. Given sufficient depth for the search ($d = 3$ was sufficient for our experiments, as our reward signal is fine-grained), STRUCT will produce two sentences joined by the conjunction "and". Again, we follow prior work in our experiment design (Koller and Petrick, 2011).

As we can see in Figures 4a, 4b, and 4c, STRUCT successfully generates results for conjunctions of up to five sentences. This is not a hard upper bound, but generation times begin to be impractically large at that point. Fortunately, human language tends toward shorter sentences than these unwieldy (but technically grammatical) sentences.

STRUCT increases in generation time both as the number of sentences increases and as the number of objects per sentences increases. We compare our results to those presented in (Koller and Petrick, 2011) for CRISP with the FF Planner. They attempted to generate sentences with three entities and failed to find a result within their 4 GB memory limit. As we can see, CRISP generates a result slightly faster than STRUCT when we are working with a single entity, but works much much slower for two entities and cannot generate results for a third entity. According to Koller's findings, this is because the search space grows by a factor of the universe size with the addition of another entity (Koller and Petrick, 2011).

5 Conclusion

We have proposed STRUCT, a general-purpose natural language generation system which is comparable to current state-of-the-art generators. STRUCT formalizes the generation problem as an MDP and applies a version of the UCT algorithm, a fast online MDP planner, to solve it. Thus, STRUCT naturally handles probabilistic grammars. We demonstrate empirically that STRUCT is anytime, comparable to existing generation-as-planning systems in certain NLG tasks, and is also capable of handling other, more complex tasks such as negated communicative goals.

Though STRUCT has many interesting properties, many directions for exploration remain. Among other things, it would be desirable to integrate STRUCT with discourse planning and dialog systems. Fortunately, reinforcement learning has already been investigated in such contexts (Lemon, 2011), indicating that an MDP-based generation procedure could be a natural fit in more complex generation systems. This is a primary direction for future work. A second direction is that, due to the nature of the approach, STRUCT is highly amenable to parallelization. None of the experiments reported here use parallelization, however, to be fair to CRISP. We plan to parallelize STRUCT in future work, to take advantage of current multicore architectures. This should obviously further reduce generation time.

STRUCT is open source and available from github.com upon request.

Acknowledgments

This work was supported in part by NSF CNS-1035602. SR was supported in part by CWRU award OSA110264. The authors are grateful to Umang Banugaria for help with the STRUCT implementation.

References

D. Bauer and A. Koller. 2010. Sentence generation as planning with probabilistic LTAG. *Proceedings of the 10th International Workshop on Tree Adjoining Grammar and Related Formalisms, New Haven, CT*.

A.L. Blum and M.L. Furst. 1997. Fast planning through planning graph analysis. *Artificial intelligence*, 90(1):281–300.

R. I. Brafman and M. Tennenholtz. 2003. R-MAX-a general polynomial time algorithm for near-optimal reinforcement learning. *Journal of Machine Learning Research*, 3:213–231.

M. Fox and D. Long. 2003. PDDL2.1: An extension to PDDL for expressing temporal planning domains. *Journal of Artificial Intelligence Research*, 20:61–124.

Jorg Hoffmann and Bernhard Nebel. 2001. The FF planning system: fast plan generation through heuristic search. *Journal of Artificial Intelligence Research*, 14(1):253–302, May.

Martin Kay. 1996. Chart generation. In *Proceedings of the 34th annual meeting on Association for Computational Linguistics*, ACL '96, pages 200–204, Stroudsburg, PA, USA. Association for Computational Linguistics.

M. Kearns, Y. Mansour, and A.Y. Ng. 1999. A sparse sampling algorithm for near-optimal planning in large Markov decision processes. In *International Joint Conference on Artificial Intelligence*, volume 16, pages 1324–1331. Lawrence Erlbaum Associates Ltd.

K. Knight and V. Hatzivassiloglou. 1995. Two-level, many-paths generation. In *Proceedings of the 33rd annual meeting on Association for Computational Linguistics*, pages 252–260. Association for Computational Linguistics.

Levente Kocsis and Csaba Szepesvari. 2006. Bandit based Monte-Carlo planning. In *Proceedings of the Seventeenth European Conference on Machine Learning*, pages 282–293. Springer.

Alexander Koller and Ronald P. A. Petrick. 2011. Experiences with planning for natural language generation. *Computational Intelligence*, 27(1):23–40.

A. Koller and M. Stone. 2007. Sentence generation as a planning problem. In *Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics*, volume 45, page 336.

I. Langkilde-Geary. 2002. An empirical verification of coverage and correctness for a general-purpose sentence generator. In *Proceedings of the 12th International Natural Language Generation Workshop*, pages 17–24.

Oliver Lemon. 2011. Learning what to say and how to say it: joint optimization of spoken dialogue management and natural language generation. *Computer Speech and Language*, 25(2):210–221.

W. Lu, H.T. Ng, and W.S. Lee. 2009. Natural language generation with tree conditional random fields. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing: Volume 1-Volume 1*, pages 400–409. Association for Computational Linguistics.

M.L. Puterman. 1994. *Markov decision processes: Discrete stochastic dynamic programming*. John Wiley & Sons, Inc.

Ehud Reiter and Robert Dale. 2000. *Building Natural Language Generation Systems*. Cambridge University Press, January.

Stuart M. Shieber. 1988. A uniform architecture for parsing and generation. In *Proceedings of the 12th conference on Computational linguistics - Volume 2*, COLING '88, pages 614–619, Stroudsburg, PA, USA. Association for Computational Linguistics.

Matthew Stone, Christine Doran, Bonnie Webber, Tonia Bleam, and Martha Palmer. 2003. Microplanning with communicative intentions: The SPUD system. *Computational Intelligence*, 19(4):311–381.

M. White and J. Baldridge. 2003. Adapting chart realization to CCG. In *Proceedings of the 9th European Workshop on Natural Language Generation*, pages 119–126.