Training a Natural Language Generator From Unaligned Data

Ondřej Dušek and **Filip Jurčíček**
Charles University in Prague, Faculty of Mathematics and Physics
Institute of Formal and Applied Linguistics
Malostranské náměstí 25, CZ-11800 Prague, Czech Republic
{odusek,jurcicek}@ufal.mff.cuni.cz

Abstract

We present a novel syntax-based natural language generation system that is trainable from unaligned pairs of input meaning representations and output sentences. It is divided into sentence planning, which incrementally builds deep-syntactic dependency trees, and surface realization. Sentence planner is based on A* search with a perceptron ranker that uses novel differing subtree updates and a simple future promise estimation; surface realization uses a rule-based pipeline from the Treex NLP toolkit.

Our first results show that training from unaligned data is feasible, the outputs of our generator are mostly fluent and relevant.

1 Introduction

We present a novel approach to natural language generation (NLG) that does not require fine-grained alignment in training data and uses deep dependency syntax for sentence plans. We include our first results on the BAGEL restaurant recommendation data set of Mairesse et al. (2010).

In our setting, the task of a natural language generator is that of converting an abstract meaning representation (MR) into a natural language utterance. This corresponds to the sentence planning and surface realization NLG stages as described by Reiter and Dale (2000). It also reflects the intended usage in a spoken dialogue system (SDS), where the NLG component is supposed to translate a system output action into a sentence. While the content planning NLG stage has been used in SDS (e.g., Rieser and Lemon (2010)), we believe that deciding upon the contents of the system's utterance is generally a task for the dialogue manager. We focus mainly on the sentence planning part in this work, and reuse an existing rule-based surface realizer to test the capabilities of the generator in an end-to-end setting.

Current NLG systems usually require a separate training data alignment step (Mairesse et al., 2010; Konstas and Lapata, 2013). Many of them use a CFG or operate in a phrase-based fashion (Angeli et al., 2010; Mairesse et al., 2010), which limits their ability to capture long-range syntactic dependencies. Our generator includes alignment learning into sentence planner training and uses deep-syntactic trees with a rule-based surface realization step, which ensures grammatical correctness of the outputs. Unlike previous approaches to trainable sentence planning (e.g., Walker et al. (2001); Stent et al. (2004)), our generator does not require a handcrafted base sentence planner.

This paper is structured as follows: in Section 2, we describe the architecture of our generator. Sections 3 and 4 then provide further details on its main components. In Section 5, we describe our experiments on the BAGEL data set, followed by an analysis of the results in Section 6. Section 7 compares our generator to previous related works and Section 8 concludes the paper.

2 Generator Architecture

Our generator (see Figure 1) operates in two stages that roughly correspond to the traditional NLG stages of sentence planning and surface realization. In the first stage, a statistical sentence planner generates deep-syntactic dependency trees from the input meaning representation. These are converted into plain text sentences in the second stage by the (mostly rule-based) surface realizer.

We use deep-syntax dependency trees to represent the *sentence plan*, i.e. the intermediate data structure between the two aforementioned stages. These are ordered dependency trees that only contain nodes for content words (nouns, full verbs, adjectives, adverbs) and coordinating conjunctions.

Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing, pages 451–461,
Beijing, China, July 26-31, 2015.

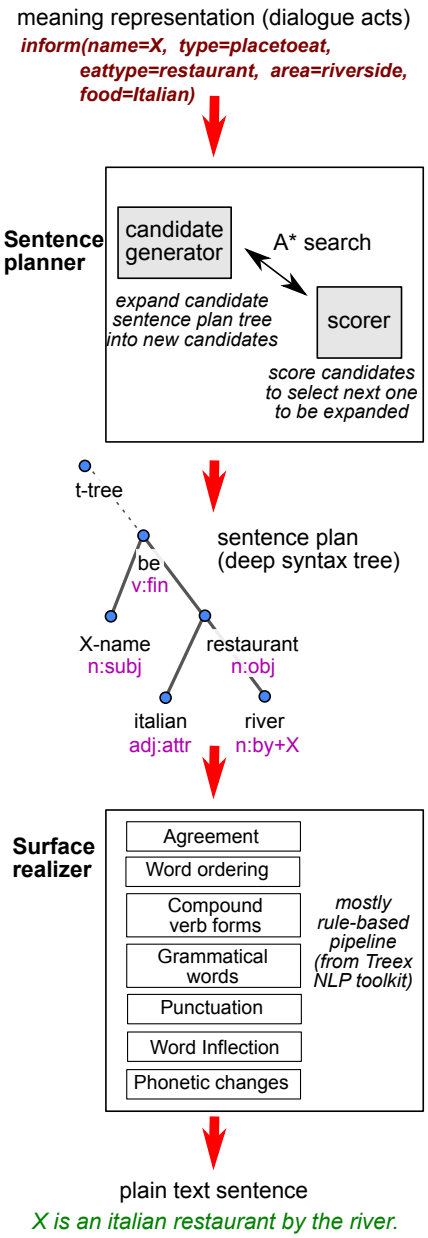

Figure 1: Overall structure of our generator

Each node has a *lemma* and a *formeme* – a concise description of its surface morphosyntactic form, which may include prepositions and/or subordinate conjunctions (Dušek et al., 2012). This structure is based on the deep-syntax trees of the Functional Generative Description (Sgall et al., 1986), but it has been simplified to fit our purposes (see Figure 1 in the middle).

There are several reasons for taking the traditional two-step approach to generation (as opposed to joint approaches, see Section 7) and using deep syntax trees as the sentence plan format: First, generating into deep syntax simplifies the task for the statistical sentence planner – the planner does not need to handle surface morphology and auxiliary words. Second, a rule-based syntactic realizer allows us to ensure grammatical correctness of the output sentences, which would be more difficult in a sequence-based and/or statistical approach.[1] And third, a rule-based surface realizer from our sentence plan format is relatively easy to implement and can be reused for any domain within the same language. As in our case, it is also possible to reuse and/or adapt an existing surface realizer (see Section 4).

Deep-syntax annotation of sentences in the training set is needed to train the sentence planner, but we assume automatic annotation and reuse an existing deep-syntactic analyzer from the Treex NLP framework (Popel and Žabokrtský, 2010).[2]

We use dialogue acts (DA) as defined in the BAGEL restaurant data set of Mairesse et al. (2010) as a MR in our experiments throughout this paper. Here, a DA consists of a dialogue act type, which is always "inform" in the set, and a list of slot-value pairs (SVPs) that contain information about a restaurant, such as food type or location (see the top of Figure 1). Our generator can be easily adapted to a different MR, though.

3 Sentence Planner

The sentence planner is based on a variant of the A* algorithm (Hart et al., 1968; Och et al., 2001; Koehn et al., 2003). It starts from an empty sentence plan tree and tries to find a path to the optimal sentence plan by iteratively adding nodes. It keeps two sets of hypotheses, i.e., candidate sentence plan trees, sorted by their score – hypotheses to expand (*open set*) and already expanded (*closed set*). It uses the following two subcomponents to guide the search:

- a *candidate generator* that is able to incrementally generate candidate sentence plan trees (see Section 3.1),
- a *scorer/ranker* that scores the appropriateness of these trees for the input MR (see Section 3.2).

[1] This issue would become more pressing in languages with richer morphology than English.

[2] See `http://ufal.mff.cuni.cz/treex`. Domain-independent deep syntax analysis for several languages is included in this framework; the English pipeline used here involves a statistical part-of-speech tagger (Spoustová et al., 2007) and a dependency parser (McDonald et al., 2005), followed by a rule-based conversion to deep syntax trees.

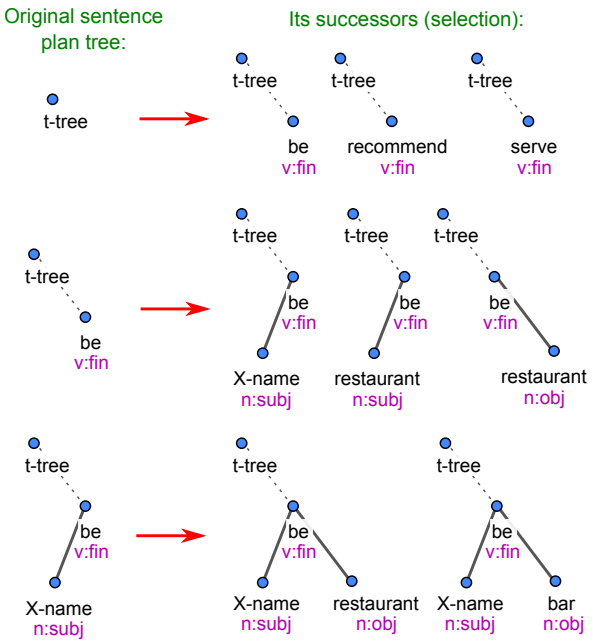

Figure 2: Candidate generator example inputs and outputs

The basic workflow of the sentence planner algorithm then looks as follows:

Init: Start from an *open set* with a single empty sentence plan tree and an empty *closed set*.

Loop:

1. Select the best-scoring candidate C from the *open set*. Add C to *closed set*.
2. The candidate generator generates $\mathbf{C}$, a set of possible successors to C. These are trees that have more nodes than C and are deemed viable. Note that $\mathbf{C}$ may be empty.
3. The scorer scores all successors in $\mathbf{C}$ and if they are not already in the *closed set*, it adds them to the *open set*.
4. Check if the best successor in the *open set* scores better than the best candidate in the *closed set*.

Stop: The algorithm finishes if the top score in the *open set* is lower than the top score in the *closed set* for d consecutive iterations, or if there are no more candidates in the *open set*. It returns the best-scoring candidate from both sets.

3.1 Generating Sentence Plan Candidates

Given a sentence plan tree, which is typically incomplete and may be even empty, the candidate generator generates its successors by adding one new node in all possible positions and with all possible lemmas and formemes (see Figure 2). While a naive implementation – trying out any combination of lemmas and formemes found in the training data – works in principle, it leads to an unmanageable number of candidate trees even for a very small domain. Therefore, we include several rules that limit the number of trees generated:

1. Lemma-formeme compatibility – only nodes with a combination of lemma and formeme seen in the training data are generated.
2. Syntactic viability – the new node must be compatible with its parent node (i.e., this combination, including the dependency left/right direction, must be seen in the training data).
3. Number of children – no node can have more children than the maximum for this lemma-formeme combination seen in the training data.
4. Tree size – the generated tree cannot have more nodes than trees seen in the training data. The same limitation applies to the individual depth levels – the training data limit the number of nodes on the n-th depth level as well as the maximum depth of any tree.

   This is further conditioned on the input SVPs – the maximums are only taken from training examples that contain the same SVPs that appear on the current input.
5. Weak semantic compatibility – we only include nodes that appear in the training data alongside the elements of the input DA, i.e., nodes that appear in training examples containing SVPs from the current input,
6. Strong semantic compatibility – for each node (lemma and formeme), we make a "compatibility list" of SVPs and slots that are present in all training data examples containing this node. We then only allow generating this node if all of them are present in the current input DA. To allow for more generalization, this rule can be applied just to lemmas

(disregarding formemes), and a certain number of SVPs/slots from the compatibility list may be required at maximum.

Only Rules 4 (partly), 5, and 6 depend on the format of the input meaning representation. Using a different MR would require changing these rules to work with atomic substructures of the new MR instead of SVPs.

While especially Rules 5 and 6 exclude a vast number of potential candidate trees, this limitation is still much weaker than using hard alignment links between the elements of the MR and the output words or phrases. It leaves enough room to generate many combinations unseen in the training data (cf. Section 6) while keeping the search space manageable. To limit the space of potential tree candidates even further, one could also use automatic alignment scores between the elements of the input MR and the tree nodes (obtained using a tool such as GIZA++ (Och and Ney, 2003)).

3.2 Scoring Sentence Plan Trees

The scorer for the individual sentence plan tree candidates is a function that maps global features from the whole sentence plan tree t and the input MR m to a real-valued score that describes the fitness of t in the context of m.

We first describe the basic version of the scorer and then our two improvements – differing subtree updates and future promise estimation.

Basic perceptron scorer

The basic scorer is based on the linear perceptron ranker of Collins and Duffy (2002), where the score is computed as a simple dot product of the features and the corresponding weight vector:

$$\text{score}(t, m) = \mathbf{w}^\top \cdot \text{feat}(t, m)$$

In the training phase, the weights $\mathbf{w}$ are initialized to one. For each input MR, the system tries to generate the best sentence plan tree given current weights, t_{top}. The score of this tree is then compared to the score of the correct gold-standard tree t_{gold}.[3] If $t_{top} \neq t_{gold}$ and the gold-standard tree ranks worse than the generated one ($\text{score}(t_{top}, m) > \text{score}(t_{gold}, m)$), the weight vector is updated by the feature value difference of the generated and the gold-standard tree:

$$\mathbf{w} = \mathbf{w} + \alpha \cdot (\text{feat}(t_{gold}, m) - \text{feat}(t_{top}, m))$$

where α is a predefined learning rate.

Differing subtree updates

In the basic version described above, the scorer is trained to score full sentence plan trees. However, it is also used to score incomplete sentence plans during the decoding. This leads to a bias towards bigger trees regardless of their fitness for the input MR. Therefore, we introduced a novel modification of the perceptron updates to improve scoring of incomplete sentence plans: In addition to updating the weights using the top-scoring candidate t_{top} and the gold-standard tree t_{gold} (see above), we also use their *differing subtrees* t^i_{top}, t^i_{gold} for additional updates.

Starting from the common subtree t_c of t_{top} and t_{gold}, pairs of differing subtrees t^i_{top}, t^i_{gold} are created by gradually adding nodes from t_{top} into t^i_{top} and from t_{gold} into t^i_{gold} (see Figure 3). To maintain the symmetry of the updates in case that the sizes of t_{top} and t_{gold} differ, more nodes may be added in one step.[4] The additional updates then look as follows:

$t^0_{top} = t^0_{gold} = t_c$
for i in $1, \ldots \min\{|t_{top}| - |t_c|, |t_{gold}| - |t_c|\} - 1$:
  $t^i_{top} = t^{i-1}_{top}$ + node(s) from t_{top}
  $t^i_{gold} = t^{i-1}_{gold}$ + node(s) from t_{gold}
  $\mathbf{w} = \mathbf{w} + \alpha \cdot (\text{feat}(t^i_{gold}, m) - \text{feat}(t^i_{top}, m))$

Future promise estimation

To further improve scoring of incomplete sentence plan trees, we incorporate a simple *future promise* estimation for the A* search intended to boost scores of sentence plans that are expected to further grow.[5] It is based on the expected number of children $E_c(n)$ of different node types (lemma-formeme pairs).[6] Given all nodes $n_1 \ldots n_{|t|}$ in a

[3] Note that the "gold-standard" sentence plan trees are actually produced by automatic annotation. For the purposes of scoring, they are, however, treated as gold standard.

[4] For example, if t_{gold} has 6 more nodes than t_c and t_{top} has 4 more, there will be 3 pairs of differing subtrees, with t^i_{gold} having 2, 4, and 5 more nodes than t_c and t^i_{top} having 1, 2, and 3 more nodes than t_c.

We have also evaluated a variant where both sets of subtrees t^i_{gold}, t^i_{top} were not equal in size, but this resulted in degraded performance.

[5] Note that this is not the same as future path cost in the original A* path search, but it plays an analogous role: weighing hypotheses of different size.

[6] $E_c(n)$ is measured as the average number of children over all occurrences of the given node type in the training data. It is expected to be domain-specific.

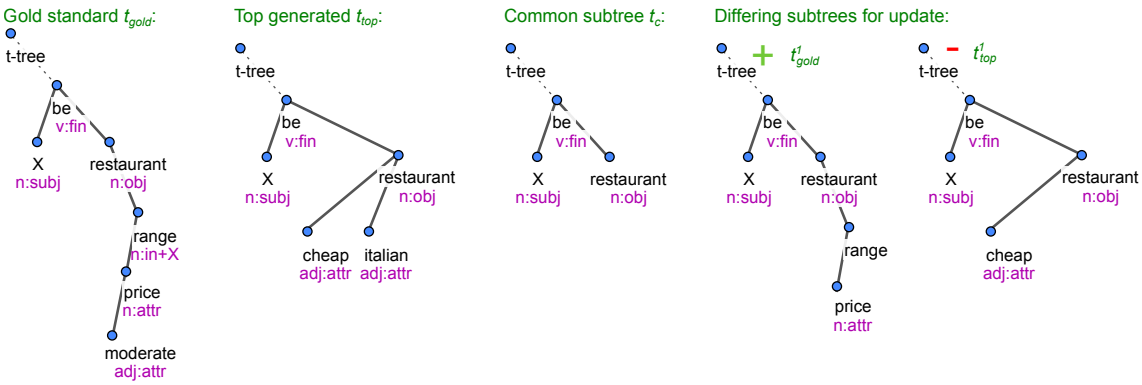

Figure 3: An example of differing subtrees

The gold standard tree t_{gold} has three more nodes than the common subtree t_c, while the top generated tree t_{top} has two more. Only one pair of differing subtrees t^1_{gold}, t^1_{top} is built, where two nodes are added into t^1_{gold} and one node into t^1_{top}.

sentence plan tree t, the future promise is computed in the following way:

$$\text{fp} = \lambda \cdot \sum \mathbf{w} \cdot \sum_{i=1}^{|t|} \max\{0, E_c(n_i) - c(n_i)\}$$

where $c(n_i)$ is the current number of children of node n_i, λ is a preset weight parameter, and $\sum \mathbf{w}$ is the sum of the current perceptron weights. Multiplying by the weights sum makes future promise values comparable to trees scores.

Future promise is added to tree scores throughout the tree generation process, but it is disregarded for the termination criterion in the *Stop* step of the generation algorithm and in perceptron weight updates.

Averaging weights and parallel training

To speed up training using parallel processing, we use the iterative parameter mixing approach of McDonald et al. (2010), where training data are split into several parts and weight updates are averaged after each pass through the training data. Following Collins (2002), we record the weights after each training pass, take an average at the end, and use this as the final weights for prediction.

4 Surface Realizer

We use the English surface realizer from the Treex NLP toolkit (cf. Section 2 and (Ptáček, 2008)). It is a simple pipeline of mostly rule-based blocks that gradually change the deep-syntactic trees into surface dependency trees, which are then linearized to sentences. It includes the following steps:

- *Agreement* – morphological attributes of some nodes are deduced based on agreement with other nodes (such as in subject-predicate agreement).
- *Word ordering* – the input trees are already ordered, so only a few rules for grammatical words are applied.
- *Compound verb forms* – additional verbal nodes are added for verbal particles (infinitive or phrasal verbs) and for compound expressions of tense, mood, and modality.
- *Grammatical words* – prepositions, subordinating conjunctions, negation particles, articles, and other grammatical words are added into the sentence.
- *Punctuation* – nodes for commas, final punctuation, quotes, and brackets are introduced.
- *Word Inflection* – words are inflected according to the information from formemes and agreement.
- *Phonetic changes* – English "a" becomes "an" based on the following word.

The realizer is designed as domain-independent and handles most English grammatical phenomena. A simple "round-trip" test – using automatic analysis with subsequent generation – reached a BLEU score (Papineni et al., 2002) of 89.79% against the original sentences on the whole BAGEL data set, showing only minor differences between the input sentence and generation output (mostly in punctuation).

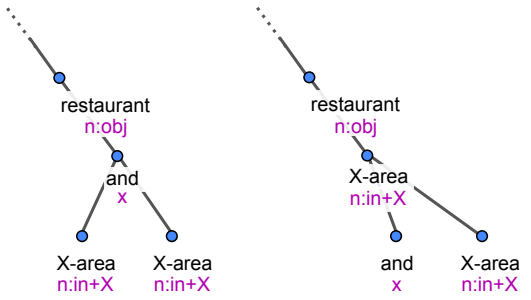

Figure 4: Coordination structures conversion: original (left) and our format (right).

5 Experimental Setup

Here we describe the data set used in our experiments, the needed preprocessing steps, and the settings of our generator specific to the data set.

5.1 Data set

We performed our experiments on the BAGEL data set of Mairesse et al. (2010), which fits our usage scenario in a spoken dialogue system and is freely available.[7] It contains a total of 404 sentences from a restaurant information domain (describing the restaurant location, food type, etc.), which correspond to 202 dialogue acts, i.e., each dialogue act has two paraphrases. Restaurant names, phone numbers, and other "non-enumerable" properties are abstracted – replaced by an "X" symbol – throughout the generation process. Note that while the data set contains alignment of source SVPs to target phrases, we do not use it in our experiments.

For sentence planner training, we automatically annotate all the sentences using the Treex deep syntactic analyzer (see Section 2). The annotation obtained from the Treex analyzer is further simplified for the sentence planner in two ways:

- Only lemmas and formemes are used in the sentence planner. Other node attributes are added in the surface realization step (see Section 5.2).
- We convert the representation of coordination structures into a format inspired by Universal Dependencies.[8] In the original Treex annotation style, the conjunction heads both conjuncts, whereas in our modification, the first conjunct is at the top, heading the coordination and the second conjunct (see Figure 4).

The coordinations can be easily converted back for the surface realizer, and the change makes the task easier for the sentence planner: it may first generate one node and then decide whether it will add a conjunction and a second conjunct.

5.2 Generator settings

In our candidate generator, we use all the limitation heuristics described in Section 3.1. For strong semantic compatibility (Rule 6), we use just lemmas and require at most 5 SVPs/slots from the lemma's compatibility list in the input DA.

We use the following feature types for our sentence planner scorer:

- current tree properties – tree depth, total number of nodes, number of repeated nodes
- tree and input DA – number of nodes per SVP and number of repeated nodes per repeated SVP,
- node features – lemma, formeme, and number of children of all nodes in the current tree, and combinations thereof,
- input features – whole SVPs (slot + value), just slots, and pairs of slots in the DA,
- combinations of node and input features,
- repeat features – occurrence of repeated lemmas and/or formemes in the current tree combined with repeated slots in the input DA,
- dependency features – parent-child pairs for lemmas and/or formemes, including and excluding their left-right order,
- sibling features – sibling pairs for lemmas and/or formemes, also combined with SVPs,
- bigram features – pairs of lemmas and/or formemes adjacent in the tree's left-right order, also combined with SVPs.

All feature values are normalized to have a mean of 0 and a standard deviation of 1, with normalization coefficients estimated from training data.

The feature set can be adapted for a different MR format – it only must capture all important parts of the MR, e.g., for a tree-like MR, the nodes and edges, and possibly combinations thereof.

[7] Available for download at: `http://farm2.user.srcf.net/research/bagel/`.

[8] `http://universaldependencies.github.io`

Setup	BLEU for training portion					NIST for training portion				
	10%	20%	30%	50%	100%	10%	20%	30%	50%	100%
Basic perc.	46.90	52.81	55.43	54.53	54.24	4.295	4.652	4.669	4.758	4.643
+ Diff-tree upd.	44.16	50.86	53.61	55.71	58.70	3.846	4.406	4.532	4.674	4.876
+ Future promise	37.25	53.57	53.80	58.15	59.89	3.331	4.549	4.607	5.071	5.231

Table 1: Evaluation on the BAGEL data set (averaged over all ten cross-validation folds)

"Training portion" denotes the percentage of the training data used in the experiment. "Basic perc." = basic perceptron updates, "+ Diff-tree upd." = with differing subtree perceptron updates, "+ Future promise" = with future promise estimation. BLEU scores are shown as percentages.

Based on our preliminary experiments, we use 100 passes over the training data and limit the number of iterations d that do not improve score to 3 for training and 4 for testing. We use a hard maximum of 200 sentence planner iterations per input DA. The learning rate α is set to 0.1. We use training data parts of 36 or 37 training examples (1/10th of the full training set) in parallel training. If future promise is used, its weight λ is set to 0.3.

The Treex English realizer expects not only lemmas and formemes, but also additional grammatical attributes for all nodes. In our experiments, we simply use the most common values found in the training data for the particular nodes as this is sufficient for our domain. In larger domains, some of these attributes may have to be also included in sentence plans.

6 Results

Same as Mairesse et al. (2010), we use 10-fold cross-validation where DAs seen at training time are never used for testing, i.e., both paraphrases or none of them are present in the full training set. We evaluate using BLEU and NIST scores (Papineni et al., 2002; Doddington, 2002) against both reference paraphrases for a given test DA.

The results of our generator are shown in Table 1, both for standard perceptron updates and our improvements – differing subtree updates and future promise estimation (see Section 3.2).

Our generator did not achieve the same performance as that of Mairesse et al. (2010) (ca. 67%).[9] However, our task is substantially harder since the generator also needs to learn the alignment of phrases to SVPs and determine whether all required information is present on the output (see also Section 7). Our differing tree updates clearly bring a substantial improvement over standard perceptron updates, and scores keep increasing with bigger amounts of training data used, whereas with plain perceptron updates, the scores stay flat. The increase with 100% is smaller since all training DAs are in fact used twice, each time with a different paraphrase.[10] A larger training set with different DAs should bring a bigger improvement. Using future promise estimation boosts the scores even further, by a smaller amount for BLEU but noticeably for NIST. Both improvements on the full training set are considered statistically significant at 95% confidence level by the paired bootstrap resampling test (Koehn, 2004). A manual inspection of a small sample of the results confirmed that the automatic scores reflect the quality of the generated sentences well.

If we look closer at the generated sentences (see Table 2), it becomes clear that the generator learns to produce meaningful utterances which mostly correspond well to the input DA. It is able to produce original paraphrases and generalizes to previously unseen DAs.

On the other hand, not all required information is always present, and some facts are sometimes repeated or irrelevant information appears. This mostly happens with input slot-value pairs that occur only rarely in the training data; we believe that a larger training set will solve this problem. Alternatively, one could introduce additional scorer features to discourage conflicting information. Another problem is posed by repeated slots in the input DA, which are sometimes not reflected properly in the generated sentence. This suggests that a further refinement of the scorer feature set may be needed.

[9] Mairesse et al. (2010) do not give a precise BLEU score number in their paper, they only show the values in a graph.

[10] We used the two paraphrases that come with each DA as independent training instances. While having two different gold-standard outputs for a single input is admittedly not ideal for a discriminative learner, it still brings an improvement in our case.

Input DA	inform(name=X-name, type=placetoeat, eattype=restaurant, near=X-near, food=Continental, food=French)
Reference	X is a French and continental restaurant near X.
Generated	X is a French and continental restaurant near X.
Input DA	inform(name=X-name, type=placetoeat, area=riverside, near=X-near, eattype=restaurant)
Reference	X restaurant is near X on the riverside.
Generated	X is a restaurant in the riverside area near X.
Input DA	inform(name=X-name, type=placetoeat, area=X-area, pricerange=moderate, eattype=restaurant)
Reference	X is a moderately priced restaurant in X.
Generated	X is a restaurant in the X area.
Input DA	inform(name=X-name, type=placetoeat, eattype=restaurant, area=riverside, food=French)
Reference	X is a French restaurant on the riverside.
Generated	X is a French restaurant in the riverside area which serves French food.
Input DA	inform(name=X-name, type=placetoeat, eattype=restaurant, pricerange=moderate, area=X-area, food=Contemporary, food=English)
Reference	X is a moderately priced English contemporary restaurant in X.
Generated	X is an English restaurant in the X area which serves expensive food in the moderate price range located in X.
Input DA	inform(name=X-name, type=placetoeat, eattype=restaurant, area=citycentre, near=X-near, food="Chinese takeaway", food=Japanese)
Reference	X is a Chinese takeaway and Japanese restaurant in the city centre near X.
Generated	X is a Japanese restaurant in the centre of town near X and X.
Input DA	inform(name=X-name, type=placetoeat, pricerange=moderate, eattype=restaurant)
Reference	X is a restaurant that offers moderate price range.
Generated	X is a restaurant in the moderate price range.

Table 2: Example generated sentences

Sentences generated when training on the full set and using differing subtree updates and future promise estimation.

7 Related Work

Previous trainable methods in sentence planning use in principle two techniques: First, in the over-generation and ranking approach (Walker et al., 2001; Stent et al., 2004), many sentence plans are generated using a rule-based planner and then the best one is selected by a statistical ranker. Second, parameter optimization trains adjustable parameters of a handcrafted generator to produce outputs with desired properties (Paiva and Evans, 2005; Mairesse and Walker, 2008). As opposed to our approach, both methods require an existing handcrafted sentence planner.

Other previous works combine sentence planning and surface realization into a single step and do not require a handcrafted base module. Wong and Mooney (2007) experiment with a phrase-based machine translation system, comparing and combining it with an inverted semantic parser based on synchronous context-free grammars. Lu et al. (2009) use tree conditional random fields over hybrid trees that combine natural language phrases with formal semantic expressions. Angeli et al. (2010) generate text from database records through a sequence of classifiers, gradually selecting database records, fields, and corresponding textual realizations to describe them. Konstas and Lapata (2013) recast the whole NLG problem as parsing over a probabilistic context-free grammar estimated from database records and their descriptions. Mairesse et al. (2010) convert input DAs into "semantic stacks", which correspond to natural language phrases and contain slots and their values on top of each other. Their generation model uses two dynamic Bayesian networks: the first one performs an ordering of the input semantic stacks, inserting intermediary stacks which correspond to grammatical phrases, the second one then produces a concrete surface realization. Dethlefs et al. (2013) approach generation as a sequence labeling task and use a conditional random field classifier, assigning a word or a phrase to each input MR element.

Unlike our work, the joint approaches typically include the alignment of input MR elements to output words in a separate preprocessing step (Wong and Mooney, 2007; Angeli et al., 2010), or require pre-aligned training data (Mairesse et al., 2010; Dethlefs et al., 2013). In addition, their basic algorithm often requires a specific input MR format, e.g., a tree (Wong and Mooney, 2007; Lu et al., 2009) or a flat database (Angeli et al., 2010; Konstas and Lapata, 2013; Mairesse et al., 2010).

While dependency-based deep syntax has been used previously in statistical NLG, the approaches known to us (Bohnet et al., 2010; Belz et al., 2012; Ballesteros et al., 2014) focus only on the surface realization step and do not include a sentence plan-

ner, whereas our work is mainly focused on statistical sentence planning and uses a rule-based realizer.

Our approach to sentence planning is most similar to Zettlemoyer and Collins (2007), which use a candidate generator and a perceptron ranker for CCG parsing. Apart from proceeding in the inverse direction and using dependency trees, we use only very generic rules in our candidate generator instead of language-specific ones, and we incorporate differing subtree updates and future promise estimation into our ranker.

8 Conclusions and Further Work

We have presented a novel natural language generator, capable of learning from unaligned pairs of input meaning representation and output utterances. It consists of a novel, A*-search-based sentence planner and a largely rule-based surface realizer from the Treex NLP toolkit. The sentence planner is, to our knowledge, first to use dependency syntax and learn alignment of semantic elements to words or phrases jointly with sentence planning.

We tested our generator on the BAGEL restaurant information data set of Mairesse et al. (2010). We have achieved very promising results, the utterances produced by our generator are mostly fluent and relevant. They did not surpass the BLEU score of the original authors; however, our task is substantially harder as our generator does not require fine-grained alignments on the input. Our novel feature of the sentence planner ranker – using differing subtrees for perceptron weight updates – has brought a significant performance improvement.

The generator source code, along with configuration files for experiments on the BAGEL data set, is available for download on Github.[11]

In future work, we plan to evaluate our generator on further domains, such as geographic information (Kate et al., 2005), weather reports (Liang et al., 2009), or flight information (Dahl et al., 1994). In order to improve the performance of our generator and remove the dependency on domain-specific features, we plan to replace the perceptron ranker with a neural network. We also want to experiment with removing the dependency on the Treex surface realizer by generating directly into dependency trees or structures into which dependency trees can be converted in a language-independent way.

[11] `https://github.com/UFAL-DSG/tgen`

Acknowledgments

This work was funded by the Ministry of Education, Youth and Sports of the Czech Republic under the grant agreement LK11221 and core research funding, SVV project 260 104, and GAUK grant 2058214 of Charles University in Prague. It used language resources stored and distributed by the LINDAT/CLARIN project of the Ministry of Education, Youth and Sports of the Czech Republic (project LM2010013).

The authors would like to thank Lukáš Žilka, Ondřej Plátek, and the anonymous reviewers for helpful comments on the draft.

References

G. Angeli, P. Liang, and D. Klein. 2010. A simple domain-independent probabilistic approach to generation. In *Proc. of the 2010 Conference on Empirical Methods in Natural Language Processing*, page 502–512.

M. Ballesteros, S. Mille, and L. Wanner. 2014. Classifiers for data-driven deep sentence generation. In *Proceedings of the 8th International Natural Language Generation Conference*, pages 108–112, Philadelphia.

A. Belz, B. Bohnet, S. Mille, L. Wanner, and M. White. 2012. The Surface Realisation Task: Recent Developments and Future Plans. In *INLG 2012*, pages 136–140.

B. Bohnet, L. Wanner, S. Mille, and A. Burga. 2010. Broad coverage multilingual deep sentence generation with a stochastic multi-level realizer. In *Proc. of the 23rd International Conference on Computational Linguistics*, page 98–106.

M. Collins and N. Duffy. 2002. New ranking algorithms for parsing and tagging: Kernels over discrete structures, and the voted perceptron. In *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, page 263–270, Stroudsburg, PA, USA. Association for Computational Linguistics.

M. Collins. 2002. Discriminative training methods for hidden Markov models: Theory and experiments with perceptron algorithms. In *Proceedings of the ACL-02 conference on Empirical methods in natural language processing-Volume 10*, page 1–8. Association for Computational Linguistics.

D. A. Dahl, M. Bates, M. Brown, W. Fisher, K. Hunicke-Smith, D. Pallett, E. Rudnicky, and E. Shriberg. 1994. Expanding the scope of the ATIS

task: the ATIS-3 corpus. In *in Proc. ARPA Human Language Technology Workshop '92, Plainsboro, NJ*, pages 43–48. Morgan Kaufmann.

N. Dethlefs, H. Hastie, H. Cuayáhuitl, and O. Lemon. 2013. Conditional Random Fields for Responsive Surface Realisation using Global Features. In *Proceedings of ACL*, Sofia.

G. Doddington. 2002. Automatic evaluation of machine translation quality using N-gram co-occurrence statistics. In *Proceedings of the Second International Conference on Human Language Technology Research*, pages 138–145, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

O. Dušek, Z. Žabokrtský, M. Popel, M. Majliš, M. Novák, and D. Mareček. 2012. Formemes in English-Czech deep syntactic MT. In *Proceedings of the Seventh Workshop on Statistical Machine Translation*, page 267–274, Montreal.

P. E. Hart, N. J. Nilsson, and B. Raphael. 1968. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107.

R. J. Kate, Y. W. Wong, and R. J. Mooney. 2005. Learning to transform natural to formal languages. In *Proceedings of the National Conference on Artificial Intelligence*, volume 20. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999.

P. Koehn, F. J. Och, and D. Marcu. 2003. Statistical phrase-based translation. In *Proceedings of NAACL-HLT - Volume 1*, page 48–54, Stroudsburg, PA, USA. Association for Computational Linguistics.

P. Koehn. 2004. Statistical significance tests for machine translation evaluation. In *Proceedings of EMNLP*, page 388–395.

I. Konstas and M. Lapata. 2013. A global model for concept-to-text generation. *Journal of Artificial Intelligence Research*, 48:305–346.

P. Liang, M. I. Jordan, and D. Klein. 2009. Learning semantic correspondences with less supervision. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP: Volume 1-Volume 1*, page 91–99.

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

F. Mairesse and M. Walker. 2008. Trainable generation of big-five personality styles through data-driven parameter estimation. In *Proc. of the 46th Annual Meeting of the ACL (ACL)*, page 165–173.

F. Mairesse, M. Gašić, F. Jurčíček, S. Keizer, B. Thomson, K. Yu, and S. Young. 2010. Phrase-based statistical language generation using graphical models and active learning. In *Proc. of the 48th Annual Meeting of the ACL*, page 1552–1561.

R. McDonald, F. Pereira, K. Ribarov, and J. Hajič. 2005. Non-projective dependency parsing using spanning tree algorithms. In *Proceedings of the conference on Human Language Technology and Empirical Methods in Natural Language Processing*, page 523–530.

R. McDonald, K. Hall, and G. Mann. 2010. Distributed training strategies for the structured perceptron. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 456–464. Association for Computational Linguistics.

F. J. Och and H. Ney. 2003. A systematic comparison of various statistical alignment models. *Computational Linguistics*, 29(1):19–51.

F. J. Och, N. Ueffing, and H. Ney. 2001. An efficient A* search algorithm for statistical machine translation. In *Proceedings of the Workshop on Data-driven Methods in Machine Translation - Volume 14*, page 1–8, Stroudsburg, PA, USA. Association for Computational Linguistics.

D. S. Paiva and R. Evans. 2005. Empirically-based control of natural language generation. In *Proc. of the 43rd Annual Meeting of ACL*, page 58–65, Stroudsburg, PA, USA. ACL.

K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu. 2002. BLEU: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*, page 311–318.

M. Popel and Z. Žabokrtský. 2010. TectoMT: modular NLP framework. In *Proceedings of IceTAL, 7th International Conference on Natural Language Processing*, page 293–304, Reykjavík.

J. Ptáček. 2008. Two tectogrammatical realizers side by side: Case of English and Czech. In *Fourth International Workshop on Human-Computer Conversation*, Bellagio, Italy.

E. Reiter and R. Dale. 2000. *Building Natural Language Generation Systems*. Cambridge Univ. Press.

V. Rieser and O. Lemon. 2010. Natural language generation as planning under uncertainty for spoken dialogue systems. In *Empirical methods in natural language generation*, page 105–120.

P. Sgall, E. Hajičová, and J. Panevová. 1986. *The meaning of the sentence in its semantic and pragmatic aspects*. D. Reidel, Dordrecht.

D. J. Spoustová, J. Hajič, J. Votrubec, P. Krbec, and P. Květoň. 2007. The Best of Two Worlds: Cooperation of Statistical and Rule-based Taggers for Czech. In *Proceedings of the Workshop on Balto-Slavonic Natural Language Processing: Information Extraction and Enabling Technologies*, pages 67–74. Association for Computational Linguistics.

A. Stent, R. Prasad, and M. Walker. 2004. Trainable sentence planning for complex information presentation in spoken dialog systems. In *Proceedings of the 42nd Annual Meeting on Association for Computational Linguistics*, pages 79–86.

M. A. Walker, O. Rambow, and M. Rogati. 2001. SPoT: a trainable sentence planner. In *Proc. of 2nd meeting of NAACL*, page 1–8, Stroudsburg, PA, USA. ACL.

Y. W. Wong and R. J. Mooney. 2007. Generation by inverting a semantic parser that uses statistical machine translation. In *Proc. of Human Language Technologies: The Conference of the North American Chapter of the ACL (NAACL-HLT-07)*, page 172–179.

L. S. Zettlemoyer and M. Collins. 2007. Online learning of relaxed CCG grammars for parsing to logical form. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 678–687, Prague.