Data-driven sentence generation with non-isomorphic trees

Miguel Ballesteros[1] **Bernd Bohnet**[2] **Simon Mille**[1] **Leo Wanner**[1,3]
[1]Natural Language Processing Group, Pompeu Fabra University, Barcelona, Spain
[2]Google Inc.
[3]Catalan Institute for Research and Advanced Studies (ICREA)
[1,3]{name.lastname}@upf.edu [2]bohnetbd@google.com

Abstract

Abstract structures from which the generation naturally starts often do not contain any functional nodes, while surface-syntactic structures or a chain of tokens in a linearized tree contain all of them. Therefore, data-driven linguistic generation needs to be able to cope with the projection between non-isomorphic structures that differ in their topology and number of nodes. So far, such a projection has been a challenge in data-driven generation and was largely avoided. We present a fully stochastic generator that is able to cope with projection between non-isomorphic structures. The generator, which starts from PropBank-like structures, consists of a cascade of SVM-classifier based submodules that map in a series of transitions the input structures onto sentences. The generator has been evaluated for English on the Penn-Treebank and for Spanish on the multi-layered Ancora-UPF corpus.

1 Introduction

Applications such as machine translation that inherently draw upon sentence generation increasingly deal with deep meaning representations; see, e.g., (Aue et al., 2004; Jones et al., 2012; Andreas et al., 2013). Deep representations tend to differ in their topology and number of nodes from the corresponding surface structures since they do not contain, e.g., any functional nodes, while syntactic structures or chains of tokens in linearized trees do. This means that sentence generation needs to be able to cope with the projection between non-isomorphic structures. However, most of the recent work in data-driven sentence generation still avoids this challenge. Some systems focus on syntactic generation (Bangalore and Rambow, 2000; Langkilde-Geary, 2002; Filippova and Strube, 2008) or linearization and inflection (Filippova and Strube, 2007; He et al., 2009; Wan et al., 2009; Guo et al., 2011a), and avoid thus the need to cope with this projection all together; some use a rule-based module to handle the projection between non-isomorphic structures (Knight and Hatzivassiloglou, 1995; Langkilde and Knight, 1998; Bohnet et al., 2011); and some adapt the meaning structures to be isomorphic with syntactic structures (Bohnet et al., 2010). However, it is obvious that a "syntacticization" of meaning structures can be only a temporary workaround and that a rule-based module raises the usual questions of coverage, maintenance and portability.

In this paper, we present a fully stochastic generator that is able to cope with the projection between non-isomorphic structures.[1] Such a generator can be used as a stand-alone application and also, e.g., in text simplification (Klebanov et al., 2004) or deep machine translation (Jones et al., 2012) (where the transfer is done at a deep level). In abstractive summarization, it facilitates the generation of the summaries, and in extractive summarization a better sentence fusion.[2]

[1]The data-driven sentence generator is available for public downloading at https://github.com/talnsoftware/deepgenerator/wiki.

[2]For all of these applications, the deep representation can be obtained by a deep parser, such as, e.g., (Ballesteros et al., 2014a).

The generator, which starts from elementary predicate-argument lexico-structural structures as used in sentence planning by Stent et al. (2004), consists of a cascade of Support Vector Machines (SVM)-classifier based submodules that map the input structures onto sentences in a series of transitions. Following the idea presented in (Ballesteros et al., 2014b), a separate SVM-classifier is defined for the mapping of each linguistic category. The generator has been tested on Spanish with the multi-layered Ancora-UPF corpus (Mille et al., 2013) and on English with an extended version of the dependency Penn TreeBank (Johansson and Nugues, 2007).

The remainder of the paper is structured as follows. In the next section, we briefly outline the fundamentals of sentence generation as we view it in our work, focusing in particular on the most challenging part of it: the transition between the non-isomorphic predicate-argument lexico-structural structures and surface-syntactic structures. Section 3 outlines the setup of our system. Section 4 discusses the experiments we carried out and the results we obtained. In Section 5, we briefly summarize related work, before in Section 6 some conclusions are drawn and future work is outlined.

2 The Fundamentals

Sentence generation realized in this paper is part of the sentence synthesis pipeline argued for by Mel'čuk (1988). It consists of a sequence of two mappings:

1. Predicate-argument lexico-structural structure → Syntactic structure
2. Syntactic Structure → Linearized structure

Following the terminology in (Mel'čuk, 1988), we refer to the predicate-argument lexico-structural structures as "deep-syntactic structures" (DSyntSs) and to the syntactic structures as "surface-syntactic structures" (SSyntSs).

While SSyntSs and linearized structures are isomorphic, the difference in the linguistic abstraction of the DSyntSs and SSyntSs leads to divergences that impede the isomorphy between the two and make the first mapping a challenge for statistical generation. Therefore, we focus in this section on the presentation of the DSyntSs and SSyntSs and the mapping between them.

2.1 DSyntSs and SSyntSs

2.1.1 Input DSyntSs

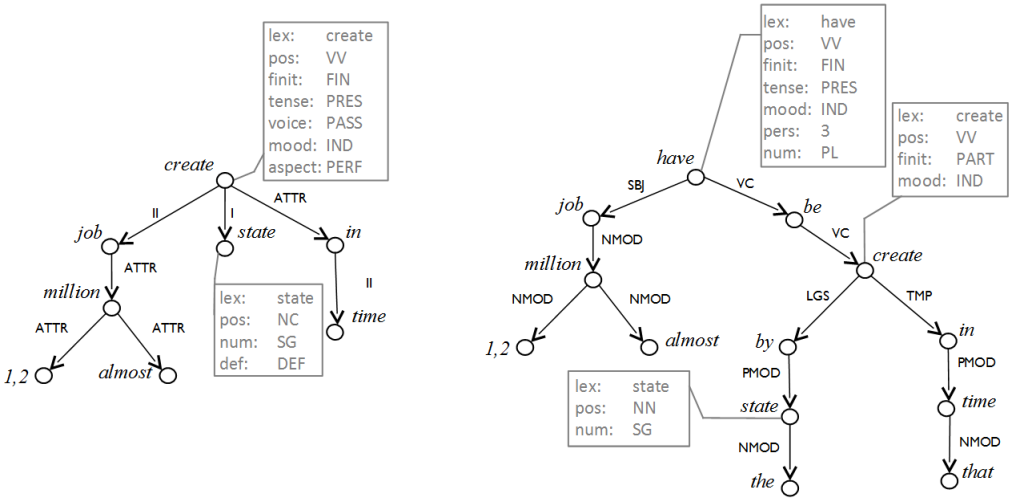
DSyntSs are very similar to the PropBank (Babko-Malaya, 2005) structures and the structures as used for the deep track of the First Surface Realization Shared Task (SRST, (Belz et al., 2011)) annotations. DSyntSs are connected trees that contain only meaning-bearing lexical items and both predicate-argument (indicated by Roman numbers: *I*, *II*, *III*, *IV*, ...) and lexico-structural, or deep-syntactic, (*ATTR(ibutive)*, *APPEND(itive)* and *COORD(inative)*) relations. In other words, they do not contain any punctuation and functional nodes, i.e., governed elements, auxiliaries and determiners. Governed elements such governed prepositions and subordinating conjunctions are dropped because they are imposed by sub-categorization restrictions of the predicative head and void of own meaning—as, for instance, *to* in *give TO your friend* or *that* in *I know that you will come*.[3] Auxiliaries do not appear as nodes in DSyntSs. Rather, the information they encode is captured in terms of tense, aspect and voice attributes of the corresponding full verbal nodes. Equally, determiners are substituted by attribute–value pairs of givenness they encode, assigned to their governors. See Figure 1 (a) for a sample DSyntS.[4]

2.1.2 SSyntSs

SSyntSs are connected dependency trees in which the nodes are labeled by open or closed class lexical items and the edges by grammatical function relations of the type 'subject', 'oblique_object', 'adverbial', 'modifier', etc. A SSyntS is thus a typical dependency tree as used in data-driven syntactic parsing (Hajič et al., 2009) and generation (Belz et al., 2011). See Figure 1 (b) for illustration of a SSyntS.

[3] In contrast, *on* in *the bottle is on the table* is not dropped because it is semantic.

[4] "That" is considered a kind of determiner (to be derived from the Information Structure). This is the reason to omit it in the deep structure.



(a) (b)

Figure 1: A DSyntS (a) and its corresponding SSyntS (b) for the sentence *Almost 1.2 million jobs have been created by the state in that time*

2.2 Projection of DSyntS to SSyntS

In order to project a DSyntS onto its corresponding SSyntS in the course of generation (where both DSyntSs and their corresponding SSyntSs are stored in the 14-column CoNLL'09 format), the following types of actions need to be performed:[5]

1. Project each node in the DSyntS onto its SSynS-correspondence. This correspondence can be a single node, as, e.g., *job* $\rightarrow$ [*NN*] (where *NN* is a noun), or a subtree (*hypernode*, known as *syntagm* in linguistics), as, e.g., *time* $\rightarrow$ [*DT NN*] (where *DT* is a determiner and *NN* a noun) or *create* $\rightarrow$ [V_{AUX} V_{AUX} *VB IN*] (where V_{AUX} is an auxiliary, *VB* a full verb and *IN* a preposition). In formal terms, we assume any SSyntS-correspondence to be a hypernode with a cardinality ≥ 1.

2. Generate the correct lemma for the nodes in SSyntS that do not have a 1:1 correspondence with an origin DSyntS node (as *DT* and V_{AUX} above).[6]

3. Establish the dependencies within the individual SSyntS-hypernodes.

4. Establish the dependencies between the SSyntS-hypernodes (more precisely, between the nodes of different SSyntS-hypernodes) to obtain a connected SSyntS-tree.

2.3 Treebanks used in the experiments

2.3.1 Spanish Treebank

For the validation of the performance of our generator on Spanish, we use the AnCora-UPF treebank, which contains only about 100,000 tokens, but which has been manually annotated and validated on the SSyntS- and DSyntS-layers, such that its quality is rather high. The deep annotation does not contain any functional prepositions since they have been removed for all predicates of the corpus, and the DSyntS-relations have been edited following annotation guidelines. AnCora-UPF SSyntSs are annotated with fine-grained dependencies organized in a hierarchical scheme (Mille et al., 2012), in a similar fashion as the dependencies of the Stanford Scheme (de Marneffe et al., 2006).[7] Thus, it is possible to use the full set of labels or to reduce it according to our needs. We performed preliminary experiments in order to assess which tag granularity is better suited for generation and came up with the 31-label tagset.

[5]For Spanish, we apply after the DSyntS–SSyntS transition in a postprocessing stage rules for the generation of relative pronouns that are implied by the the SSyntS. Since we cannot count on the annotation of coreference in the training data, we do not treat other types of referring expressions.

[6]The lemmas of nodes with 1:1 correspondence are the same in both structures.

[7]The main difference with the Stanford scheme is that in AnCora-UPF no distinction is explicitly made between argumental and non-argumental dependencies.

2.3.2 English Treebank

For the validation of the generator on English, we use the dependency Penn TreeBank (about 1,000,000 tokens), which we extend by a DSynt layer defined by the same deep dependency relations, features and node correspondences as the Spanish DSynt layer. The Penn TreeBank DSynt layer is obtained by a rule-based graph transducer. The transducer removes definite and indefinite determiners, auxiliaries, THAT complementizers, TO infinitive markers, and a finite list of functional prepositions. The functional prepositions have been manually compiled from the description and examples of the roles in the PropBank and NomBank annotations of the 150 most frequent predicates of the corpus. A dictionary has been built, which contains for each of the 150 predicates the argument slots (roles) and the prepositions associated to it, such that given a predicate and a preposition, we know to which role it corresponds. Consider, for illustration, Figure 2, which indicates that for the nominal predicate *plan_01*, a dependent introduced by the preposition *to* corresponds to the second argument of *plan_01*, while a dependent introduced by *for* is its third argument.

```
plan_01 : predicate_ {
 NN = {
  to = {DeepRel = II}
  for = {DeepRel = III}
 }}
```

Figure 2: A sample (partial) mapping dictionary entry

For each possible surface dependency relation between a governor and a dependent, a default mapping is provided, which is applied if

(i) The syntactic structure fulfills the conditions of the default mapping (e.g., '*subject*' is by default mapped onto '*I*' unless it is the subject of a passive verb, in which case it is mapped to the second argument '*II*'), and

(ii) The pair governor–dependent is not found in the dictionary; that is, if the dependent of the SSyntS dependency relation is a preposition found in the governor's entry in the dictionary, the information provided in the dictionary is used instead of the default mapping.[8] For instance, in the sentence *Sony announced its plans to hire Mr. Guber*, *to* is a dependent of *plan* with the SSyntS dependency *NMOD*. *NMOD* is by default mapped onto the deep relation *ATTR*, but since in the dictionary entry of *plan* it is stated that a dependent introduced by *to* is mapped to 'II' (cf. Figure 2), *II* is the relation that appears in the DSyntS-annotation.

The features *definiteness*, *voice*, *tense*, *aspect* in the *FEATS* column of the CoNLL format capture the information conveyed by determiners and auxiliaries. The conversion procedure maps surface dependency relations as found in the Penn TreeBank onto the restricted set of deep dependency relations as described in Section 2.1.1.

The nodes in the original (surface-oriented) and deep annotations are connected through their IDs. In the *FEATS* column of the output CoNLL file, *id0* indicates the deep identifier of a word, while *id1* indicates the ID of the surface node it corresponds to. There are less nodes in DSyntSs than in SSyntSs since SSyntSs contain all the words of a sentence. Hence, a DSynt-node can correspond to several SSyntS nodes. Multiple correspondences are indicated by the presence of the *id2* (*id3*, *id4*, etc) feature in the *FEATS* column.

3 Deep Generation

3.1 Baselines

Since no available data-driven generator uses as input DSyntSs, we developed as baselines two rule-based graph transducer generators which produce for English respectively Spanish the best possible SSyntSs, using only the information contained in the starting DSyntS.

The two baseline generators are structured similarly: both contain around 50 graph transducer rules, separated into two clusters. The first cluster maps DSyntS-nodes onto SSyntS-nodes, while the second one handles the introduction of SSyntS dependency relations between the generated SSyntS-nodes. For instance, in English, one rule maps DSyntS-nodes that have a one-to-one correspondence in the SSyntS

[8] In the PropBank annotation, a distinction is made between external and internal arguments, such that for some predicates the arguments are numbered starting from '0', and for other starting from '1'. This has been normalized in order to make all arguments start from '1' for all predicates.

if N_1 is a V_{fin} **and** (($R_{1,2}$ == I **and** N_1 is in active voice **and** N_2 is **not** *by*)
or ($R_{1,2}$ == II **and** N_1 is in passive voice))
  if ∃ one-to-one correspondence between N_{D_i} **and** N_{S_i}
    then introduce *SBJ* between N_{S_1} **and** N_{S_2}
  else
    if N_{S_2} is top node of the SSyntS hypernode **and**
    ((N_{S_1} is top node of the SSynt hypernode **and** is AUX)
     or (N_{S_1} is the bottom node of the SSynt hypernode **and** is V_{fin})
     or (N_{S_1} is **not** top node **or** bottom node of the SSynt-hypernode **and** is AUX))
     then introduce *SBJ* between N_{S_1} **and** N_{S_2}
    endif
endif

Figure 3: Sample graph transducer rule

(simple nouns, verbs, adverbs, adjectives, etc.), 22 rules map DSyntS-nodes that have a one-to-many correspondence in the SSyntS (N → DET+NN, N → DET+NN+governed PREP, V → AUX+VV, V → *that* COMPL+AUX+VV+governed PREP, etc.), and 25 rules generate the dependency relations.[9] The transduction rules apply in two phases, see Figure 3. During the first phase, all nodes and intra-hypernode dependencies are created in the output structure. During the second phase, all inter-hypernode dependencies are established. Since there are one-to-many DSyntS-SSyntS correspondences, the rules of the second phase have to ensure that the correct output nodes are targeted, i.e., that *jobs* in Figure 1(b) is made a dependent of *have*, and not of *been* or *created*, which all correspond to *create* in the input. Consider, for illustration of the complexity of the rule-based generator, the transduction rule in Figure 3. The rule creates the SSynt dependency relation *SBJ* (*subject*) in a target SSyntS (with a governor node N_{D_1} and a dependent node N_{D_2} linked by a deep dependency relation $R_{1,2}$ in the input DSyntS and two nodes N_{S_1} and N_{S_2} which correspond to N_{D_1} and N_{D_2} respectively in the target SSyntS).

The evaluation shows that all straightforward mappings are performed correctly; English auxiliaries, *that* complementizers, infinitive markers and determiners are introduced, and so are Spanish auxiliaries, reflexive pronouns, and determiners. That is, the rules produce well-formed SSyntSs of all possible combinations of auxiliaries, conjunctions and/or prepositions for verbs, determiners and/or prepositions for nouns, adjectives and adverbs.

When there are several possible mappings, the baseline takes decisions by default. For example, when a governed preposition must be introduced, we always introduce the most common one (*of* in English, *de* 'of' in Spanish).

3.2 Data-Driven Generator

The data-driven generator is defined as a tree transducer framework that consists of a cascade of 6 data-driven small tasks; cf. Figure 4. The first four tasks capture the actions 1.–4. from Section 2.2; the 5th linearizes the obtained SSyntS. Figure 4 provides a sample input and output of each submodule. The system outputs a 14 column CoNLL'09 linearized format without morphological inflections or punctuation marks.

In the next sections, we discuss how these actions are realized and how they are embedded into the overall generation process.

The intra- and inter-hypernode dependency determination works as an informed dependency parser that uses the DSyntS as input. The search space is thus completely pruned. Note also that for each step, the space of classes for the SVMs is based on linguistic facts extracted from the training corpus (for instance, for the preposition generation SVM, the classes are the possible prepositions; for the auxiliary generation SVM, the possible auxiliaries, etc.).

3.2.1 Hypernode Identification

Given a node n_d from the DSyntS, the system must find the shape of the surface hypernode that corresponds to n_d in the SSyntS. The hypernode identification SVMs use the following features:

> PoS of n_d, PoS of n_d's head, verbal voice (*active*, *passive*) and aspect (*perfective*, *progressive*) of the current node, lemma of n_d, and n_d's dependencies.

In order to simplify the task, we define the shape of a surface hypernode as a list of surface PoS tags. This unordered list contains the PoS of each of the lemmas contained within the hypernode and a tag that encodes the original deep node; for instance:

[9] 'N' stands for "noun", 'NN' for "common noun", 'DET' for "determiner", 'PREP' for "preposition", 'V' for "verb", 'AUX' for "auxiliary verb", 'VV' for "main verb", and 'COMPL' for "complementizer".

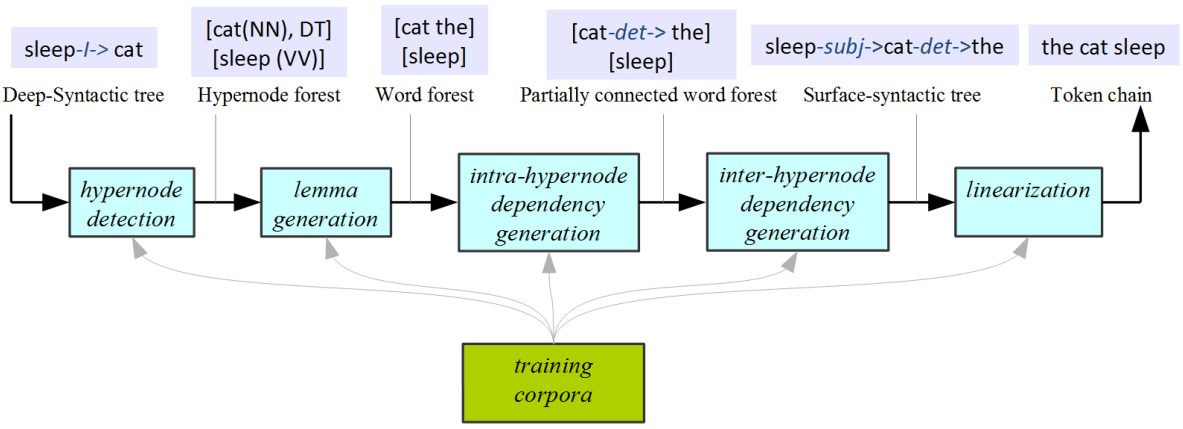


Figure 4: Workflow of the Data-Driven Generator.

[NN(deep), DT]

For each deep, i.e., DSyntS, PoS tag (which can be one of the following four: *N* (noun), *V* (verb), *Adv* (adverb), *A* (adjective)), a separate multi-class classifier is defined.[10] For instance, in the case of *N*, the N-classifier will use the above features to assign to the a DSynt-node with PoS *N* the most appropriate (most likely) hypernode—in this case, [*NN*(deep), *DT*].

3.2.2 Lemma Generation

Once the hypernodes of the SSyntS under construction have been produced, the functional nodes that have been newly introduced in the hypernodes must be assigned a lemma label. The lemma generation SVMs use the following features of the deep nodes n_d in the hypernodes to select the most likely lemma:

> verbal finiteness (*finite*, *infinitive*, *gerund*, *participle*) and aspect (*perfective*, *progressive*), degree of definiteness of nouns, PoS of n_d, lemma of n_d, PoS of the head of n_d

Again, for each surface PoS tag, a separate classifier is defined. Thus, the DT-classifier would pick for the hypernode [*NN*(deep), *DT*] the most likely lemma for the *DT*-node (optimally, a determiner).

[10] As will be seen in the discussion of the results, the strategy proposed by Ballesteros et al. (2014b) to define a separate classifier for each linguistic category here and in the other stages largely pays off because it reduces the classification search space enormously and thus leads to a higher accuracy.

3.2.3 Intra-hypernode Dependency Generation

Given a hypernode and its lemmas provided by the two previous stages, the dependencies (i.e., the dependency attachments and dependency labels) between the elements of the created SSyntS hypernodes must be determined (and thus also the governors of the hypernodes). For this task, the intra-hypernode dependency generation SVMs use the following features:

> lemmas included in the hypernode, PoS-tags of the lemmas in the hypernode, *voice* of the head h of the hypernode, deep dependency relation to h.

For each kind of hypernode, dynamically a separate classifier is generated.[11] In the case of the hypernode [*NN*(deep), *DT*], the corresponding classifier will create a link between the determiner and the noun, with the noun as head and the determiner as dependent because it is the best link that it can find; cf. Figure 5 for illustration. We ensure that the output of the classifiers is a tree by controlling that every node (except the root) has one and only one governor. The DSynt input is a tree; in the case of hypernodes of cardinality one, the governor/dependent relation is maintained; in the case of hypernodes of higher cardinality, only one node receives an incoming arc and only one can govern another hypernode.

3.2.4 Inter-hypernode Dependency Generation

Once the individual hypernodes have been converted into connected dependency subtrees, the hy-

[11] This implies that the number of classifiers varies depending on the training set. For instance, during the intra-hypernode dependency creation for Spanish, 108 SVMs are generated.

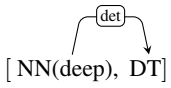


Figure 5: Internal dependency within a hypernode

pernodes must be connected between each other, such that we obtain a complete SSyntS. The inter-hypernode dependency generation SVMs use the following features of a hypernode s_s to determine for each hypernode its governor. For each hypernode with a distinct internal dependency pattern, a separate classifier is dynamically derived (for our treebanks, we obtained 114 different SVM classifiers because they also take into account hypernodes with just one token).:

> the internal dependencies of s_s, the head of s_s, the lemmas of s_s, the PoS of the dependent of the head of s_s in DSyntS

For instance, the classifier for the hypernode [*JJ*(deep)] is most likely to identify as its governor *NN* in the hypernode [*NN*(deep), *DT*]; cf. Figure 6.

The task faced by the inter-hypernode dependency classifiers is the same as that of a dependency parser, only that its search space is very small (which is favorably reflected in the accuracy figures).

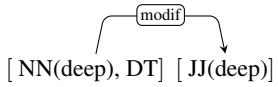


Figure 6: Surface dependencies between two hypernodes.

3.3 Linearization

Once we obtained a SSyntS, the linearizer must find the correct order of the words. There is already a body of work available on statistical linearization. Therefore, these tasks were not in the focus of our work. Rather, we adopt the most successful technique of the first SRST (Belz et al., 2011), a bottom-up tree linearizer that orders bottom-up each head and its children (Bohnet et al., 2011; Guo et al., 2011a). This has the advantage that the linear order obtained previously can provide context features for ordering sub-trees higher up in the dependency tree. Each head and its children are ordered with a beam search.

The beam is initialized with entries of single words that are expanded in the next step by the remaining words of the sub-tree, which results in a number of new entries for the next iteration. After the expansion step, the new beam entries are sorted and pruned. We keep the 30 best entries and continue with the expansion and pruning steps until no further nodes of the subtree are left. We take an SVM to obtain the scores for sorting the beam entries, using the same feature templates as in Guo et al. (2011b) and Bohnet et al. (2011).

4 Experiments and Results

In our experiments, the Spanish treebank has been divided into: (i) a development set of 219 sentences, with 3,437 tokens in the DSyntS treebank and 4,799 tokens in the SSyntS treebank (with an average of 21.91 words by sentence in SSynt); (ii) a training set of 3,036 sentences, with 57,665 tokens in the DSyntS treebank and 84,668 tokens in the SSyntS treebank (with an average of 27.89 words by sentence in SSynt); and a (iii) a held-out test for evaluation of 258 sentences, with 5,878 tokens in the DSyntS treebank and 8,731 tokens in the SSyntS treebank (with an average of 33.84 words by sentence in SSynt).

For the English treebank, we used a classical split of (i) a training set of 39,279 sentences, with 724,828 tokens in the DSynt treebank and 958,167 tokens in the SSynt treebank (with an average of 24.39 words by sentence in SSynt); and (ii) a test set of 2,399 sentences, with 43,245 tokens in the DSynt treebank and 57,676 tokens in the SSynt treebank (with an average of 24.04 words by sentence in SSynt).

In what follows, we show the system performance on both treebanks. The Spanish treebank was used for development and testing, while the English treebank was only used for testing.

4.1 Results

In this section, we present the performance of, first of all, the individual tasks of the data-driven DSyntS–SSyntS projection, since these have been the challenging tasks that we addressed. Table 1 shows similar results for all tasks on the development and test sets with gold-standard input, that is,

the results of the classifiers as a stand-alone module, assuming that the previous module provides a perfect output.

Spanish Dev.set	#	%
Hypernode identification	3327/3437	96.80
Lemma generation	724/767	94.39
Intra-hypernode dep. generation	756/756	100.00
Inter-hypernode dep. generation	2628/2931	89.66
Spanish Test set	#	%
Hyper-node identification	5640/5878	95.95
Lemma generation	1556/1640	94.88
Intra-hypernode dep. generation	1622/1622	100.00
Inter-hypernode dep. generation	4572/5029	90.91

Table 1: Results of the evaluation of the SVMs for the non-isomorphic transition for the Spanish DSyntS development and test sets

English Test set	#	%
Hyper-node identification	42103/43245	97.36
Lemma generation	6726/7199	93.43
Intra-hypernode dep. generation	6754/7179	94.08
Inter-hypernode dep. generation	35922/40699	88.26

Table 2: Results of the evaluation of the SVMs for the non-isomorphic transition for the English DSyntS test set

To have the entire generation pipeline in place, we carried out several linearization experiments, starting from: (i) the SSyntS gold standard, (ii) SSyntSs generated by the rule-based baselines, and (iii) SSyntSs generated by the data-driven deep generator; cf. *surface gen.*, *baseline deep gen*, and *deep gen.* respectively in Tables 3 and 4).[12]

Development Set	BLEU	NIST	Exact
surface gen.	0.754	11.29	24.20 %
baseline deep gen.	0.547	9.98	10.96 %
deep gen.	**0.582**	10.78	12.33 %
Test Set	BLEU	NIST	Exact
surface gen.	0.762	12.08	15.89 %
baseline deep gen.	0.515	10.60	2.33 %
deep gen.	**0.542**	11.24	3.49 %

Table 3: Overview of the results on the Spanish development and test sets excluding punctuation marks after the linearization

[12]Following (Langkilde-Geary, 2002; Belz et al., 2011) and other works on statistical text generation, we access the quality of the linearization module via BLEU score, NIST and exactly matched sentences.

Test Set	BLEU	NIST	Exact
surface gen.	0.91	15.26	56.02 %
baseline deep gen.	0.69	13.71	12.38 %
deep gen.	**0.77**	14.42	21.05 %

Table 4: Overview of the results on the English test set excluding punctuation marks after the linearization

4.2 Discussion and Error Analysis

In general, Tables 1–4 show that the quality of the presented deep data-driven generator is rather good both during the individual stages of the DSyntS–SSyntS transition and as part of the DSyntS–linearized sentence pipeline.

Two main problems impede an even better performance figures than those reflected in Tables 1 and 2. First, the introduction of prepositions causes most errors in hypernode detection and lemma generation: when a preposition should be introduced or not and which preposition should be introduced depends exclusively on the subcategorization frame of the governor of the DSyntS node. A corpus of a limited size does not capture the subcategorization frames of ALL predicates. This is especially true for our Spanish treebank, which is particularly small. Second, the inter-hypernode dependency suffers from the fact that the SSyntS tagset is quite fine-grained, at least in the case of Spanish, which makes the task of the classifiers harder (e.g., there are nine different types of verbal objects). In spite of these problems, each set of classifiers achieves results above 88% on the test sets.

The results of deep generation in Tables 3 and 4 can be explained by the fact of error propagation: while (only) about 1 out of 10 hypernodes and about 1 out of 10 lemmas are not correct and very little information is lost in the stage of the intra-hypernode dependencies determination, already almost 1.75 out of 10 inter-hypernode dependencies, and finally 1 out 10 linear orderings are incorrect for English and more than 2 out 10 for Spanish.

As already mentioned above, the size of the training corpus strongly affects the results. Thus, for English, for which the size of the training dataset has been 10 times bigger than for Spanish, the data-driven generator provides, without any tuning, more than 0.2 BLEU points more that for Spanish. A bigger corpus also covers more linguistic phenomena

(lexical features, subcategorization frames, syntactic sentential constructions, etc.)—which can be also exploited for rule-based generation.

The linearizer also suffers from a small size of the training set. Thus, while the small Spanish training corpus leads to 0.754 BLEU and 0.762 BLEU for the development and test sets respectively, for English, we achieve 0.91 BLEU, which is a very competitive outcome compared to other English linearizers (Song et al., 2014).

We also found that the data-driven generator tends to output slightly shorter sentences, when compared to the rule-based baseline. It is always difficult to find the best evaluation metric for plain text sentences (Smith et al., 2014). In our experiments, we used BLEU, NIST and the exact match metric. BLEU is the average of n-gram precisions and includes a brevity penalty, which reduces the score if the length of the output sentence is shorter than the gold. In other words, BLEU favors longer sentences. We believe that this is one of the reasons why the machine-learning based generator shows a bigger difference for the English test set and the Spanish development set than the rule-based baseline. Firstly, there are extremely long sentences in the Spanish test set (31 words per sentence, in the average; the longest being 165 words). Secondly, the English sentences and the Spanish development sentences are much shorter than the Spanish test sentences, such that the ML approach has the potential to perform better.

5 Related work

There is an increasing amount of work on statistical sentence generation, although hardly any addresses the problem of deep generation from semantic structures that are not isomorphic with syntactic- structures as a purely data-driven problem (as we do). To the best of our knowledge, the only exception is our earlier work in (Ballesteros et al., 2014b), where we discuss the principles of classifiers for data-driven generators. As already mentioned in Section 1, most of the state-of-the-art work focuses on syntactic generation; see, among others (Bangalore and Rambow, 2000; Langkilde-Geary, 2002; Filippova and Strube, 2008), or only on linearization and inflection (Filippova and Strube, 2007; He et al., 2009; Wan et al., 2009; Guo et al., 2011a). A number of proposals are hybrid in that they combine statistical machine learning-based generation with rule-based generation. Thus, some combine machine learning with pre-generated elements, as, e.g., (Marciniak and Strube, 2004; Wong and Mooney, 2007; Mairesse et al., 2010), or with handcrafted rules, as, e.g., (Ringger et al., 2004; Belz, 2005). Others derive automatically grammars for rule-based generation modules from annotated data, which can be used for surface generation, as, e.g., (Knight and Hatzivassiloglou, 1995; Langkilde and Knight, 1998; Oh and Rudnicky, 2002; Zhong and Stent, 2005; Bohnet et al., 2011; Rajkumar et al., 2011) or for generation from ontology triples, as, e.g., (Gyawali and Gardent, 2013).

6 Conclusions

We presented a statistical deep sentence generator that successfully handles the non-isomorphism between meaning representations and syntactic structures in terms of a principled machine learning approach. This generator has been successfully tested on an English and a Spanish corpus, as a stand-alone DSyntS–SSyntS generator and as a part of the generation pipeline. We are currently about to apply it to other languages—including Chinese, French and German. Furthermore, resources are compiled to use it for generation of spoken discourse in Arabic, Polish and Turkish.

We believe that our generator can be used not only in generation *per se*, but also, e.g., in machine translation (MT), since MT could profit from using meaning representations such as DSyntSs, which abstract away from the surface syntactic idiosyncrasies of each language, but are still linguistically motivated, as transfer representations.

Acknowledgments

Our work on deep stochastic sentence generation is partially supported by the European Commission under the contract numbers FP7-ICT-610411 (project MULTISENSOR) and H2020-RIA-645012 (project KRISTINA).

References

J. Andreas, A. Vlachos, and S. Clark. 2013. Semantic Parsing as Machine Translation. In *Proceedings of ACL '13*.

A. Aue, A. Menezes, R. Moore, C. Quirk, and E. Ringger. 2004. Statistical Machine Translation Using Labeled Semantic Dependency Graphs. In *Proceedings of TMI '04*.

Olga Babko-Malaya, 2005. *Propbank Annotation Guidelines*.

Miguel Ballesteros, Bernd Bohnet, Simon Mille, and Leo Wanner. 2014a. Deep-syntactic parsing. In *Proceedings of COLING'14*, Dublin, Ireland.

Miguel Ballesteros, Simon Mille, and Leo Wanner. 2014b. Classifiers for Data-Driven Deep Sentence Generation. In *Proceedings of the International Conference of Natural Language Generation (INLG)*.

Srinivas Bangalore and Owen Rambow. 2000. Exploiting a probabilistic hierarchical model for generation. In *Proceedings of the 18th conference on Computational linguistics-Volume 1*, pages 42–48. Association for Computational Linguistics.

Anja Belz, Mike White, Dominic Espinosa, Eric Kow, Deirdre Hogan, and Amanda Stent. 2011. The first surface realisation shared task: Overview and evaluation results. In *Proceedings of the Generation Challenges Session at the 13th European Workshop on Natural Language Generation*, pages 217–226.

Anja Belz. 2005. Statistical generation: Three methods compared and evaluated. In *Proceedings of the 10th European Workshop on Natural Language Generation*, pages 15–23.

Bernd Bohnet, Leo Wanner, Simon Mille, and Alicia Burga. 2010. Broad coverage multilingual deep sentence generation with a stochastic multi-level realizer. In *Proceedings of COLING '10*, pages "98–106".

Bernd Bohnet, Simon Mille, Benoît Favre, and Leo Wanner. 2011. StuMaBa: From deep representation to surface. In *Proceedings of ENLG 2011, Surface-Generation Shared Task*, Nancy, France.

Marie-Catherine de Marneffe, Bill MacCartney, and Christopher D Manning. 2006. Generating typed dependency parses from phrase structure parses. In *Proceedings of the 5th International Conference on Language Resources and Evaluation (LREC)*, volume 6, pages 449–454.

Katja Filippova and Michael Strube. 2007. Generating constituent order in german clauses. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, volume 45, page 320.

Katja Filippova and Michael Strube. 2008. Sentence fusion via dependency graph compression. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, EMNLP '08, pages 177–185, Stroudsburg, PA, USA. Association for Computational Linguistics.

Yuqing Guo, Deirdre Hogan, and Josef van Genabith. 2011a. Dcu at generation challenges 2011 surface realisation track. In *Proceedings of the Generation Challenges Session at the 13th European Workshop on Natural Language Generation*, pages 227–229, Nancy, France, September. Association for Computational Linguistics.

Yuqing Guo, Haifeng Wang, and Josef Van Genabith. 2011b. Dependency-based n-gram models for general purpose sentence realisation. *Natural Language Engineering*, 17(04):455–483.

B. Gyawali and C. Gardent. 2013. LOR-KBGEN, A Hybrid Approach To Generating from the KBGen Knowledge-Base. In *Proceedings of the KBGen Challenge http://www.kbgen.org/papers/*.

Jan Hajič, Massimiliano Ciaramita, Richard Johansson, Daisuke Kawahara, Maria Antònia Martí, Lluís Màrquez, Adam Meyers, Joakim Nivre, Sebastian Padó, Jan Štěpánek, Pavel Straňák, Mihai Surdeanu, Nianwen Xue, and Yi Zhang. 2009. The CoNLL-2009 shared task: Syntactic and semantic dependencies in multiple languages. In *Proceedings of the 13th Conference on Computational Natural Language Learning (CoNLL-2009), June 4-5*, Boulder, Colorado, USA.

Wei He, Haifeng Wang, Yuqing Guo, and Ting Liu. 2009. Dependency based chinese sentence realization. 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 2-Volume 2*, pages 809–816. Association for Computational Linguistics.

Richard Johansson and Pierre Nugues. 2007. Extended constituent-to-dependency conversion for English. In *Proceedings of the 16th Nordic Conference of Computational Linguistics (NODALIDA)*, pages 105–112, Tartu, Estonia, May 25-26.

B. Jones, J. Andreas, D. Bauer, K.M. Hermann, and K. Knight. 2012. Semantics-Based Machine Translation with Hyperedge Replacement Grammars. In *Proceedings of COLING '12*.

Beata Beigman Klebanov, Kevin Knight, and Daniel Marcu. 2004. Text simplification for information-seeking applications. In *On the Move to Meaningful Internet Systems, Lecture Notes in Computer Science*, pages 735–747. Springer Verlag.

Kevin Knight and Vasileios 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.

I. Langkilde and K. Knight. 1998. Generation that exploits corpus-based statistical knowledge. In *Proceedings of the COLING/ACL*, pages 704–710.

Irene 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. Citeseer.

François Mairesse, Milica Gašić, Filip Jurčíček, Simon Keizer, Blaise Thomson, Kai Yu, and Steve Young. 2010. Phrase-based statistical language generation using graphical models and active learning. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 1552–1561. Association for Computational Linguistics.

Tomasz Marciniak and Michael Strube. 2004. Classification-based generation using tag. In *Natural Language Generation*, pages 100–109. Springer.

Igor Mel'čuk. 1988. *Dependency Syntax: Theory and Practice*. State University of New York Press, Albany.

Simon Mille, Alicia Burga, Gabriela Ferraro, and Leo Wanner. 2012. How does the granularity of an annotation scheme influence dependency parsing performance? In *Proceedings of COLING 2012*, pages 839–852, Mumbai, India.

Simon Mille, Alicia Burga, and Leo Wanner. 2013. Ancora-upf: A multi-level annotation of spanish. In *Proceedings of the Second International Conference on Dependency Linguistics*, Prague, Czech Republic.

Alice H Oh and Alexander I Rudnicky. 2002. Stochastic natural language generation for spoken dialog systems. *Computer Speech & Language*, 16(3):387–407.

Rajakrishnan Rajkumar, Dominic Espinosa, and Michael White. 2011. The osu system for surface realization at generation challenges 2011. In *Proceedings of the 13th European workshop on natural language generation*, pages 236–238. Association for Computational Linguistics.

Eric Ringger, Michael Gamon, Robert C Moore, David Rojas, Martine Smets, and Simon Corston-Oliver. 2004. Linguistically informed statistical models of constituent structure for ordering in sentence realization. In *Proceedings of the 20th international conference on Computational Linguistics*, page 673. Association for Computational Linguistics.

Aaron Smith, Christian Hardmeier, and Jörg Tiedemann. 2014. Bleu is not the colour: How optimising bleu reduces translation quality.

Linfeng Song, Yue Zhang, Kai Song, and Qun Liu. 2014. Joint morphological generation and syntactic linearization. In *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence, July 27 - 31, 2014, Québec City, Québec, Canada.*, pages 1522–1528.

A. Stent, R. Prasad, and M. Walker. 2004. Trainable sentence planning for complex information presentation in spoken dialog systems. In *Proceedings of the ACL '04*, pages 79–86.

Stephen Wan, Mark Dras, Robert Dale, and Cécile Paris. 2009. Improving grammaticality in statistical sentence generation: Introducing a dependency spanning tree algorithm with an argument satisfaction model. In *Proceedings of the 12th Conference of the European Chapter of the Association for Computational Linguistics*, pages 852–860. Association for Computational Linguistics.

Yuk Wah Wong and Raymond J Mooney. 2007. Generation by inverting a semantic parser that uses statistical machine translation. In *HLT-NAACL*, pages 172–179.

Huayan Zhong and Amanda Stent. 2005. Building surface realizers automatically from corpora. *Proceedings of UCNLG*, 5:49–54.