Broad Coverage Multilingual Deep Sentence Generation with a Stochastic Multi-Level Realizer

Bernd Bohnet[1], **Leo Wanner**[1,2], **Simon Mille**[1], **Alicia Burga**[1]
[1]Department of Information and Communication Technologies
Pompeu Fabra University
[2]Institució Catalana de Recerca i Estudis Avançats (ICREA)
{first-name.last-name}@upf.edu

Abstract

Most of the known stochastic sentence generators use syntactically annotated corpora, performing the projection to the surface in one stage. However, in full-fledged text generation, sentence realization usually starts from semantic (predicate-argument) structures. To be able to deal with semantic structures, stochastic generators require semantically annotated, or, even better, multilevel annotated corpora. Only then can they deal with such crucial generation issues as sentence planning, linearization and morphologization. Multilevel annotated corpora are increasingly available for multiple languages. We take advantage of them and propose a multilingual deep stochastic sentence realizer that mirrors the state-of-the-art research in semantic parsing. The realizer uses an SVM learning algorithm. For each pair of adjacent levels of annotation, a separate decoder is defined. So far, we evaluated the realizer for Chinese, English, German, and Spanish.

1 Introduction

Recent years saw a significant increase of interest in corpus-based natural language generation (NLG), and, in particular, in corpus-based (or stochastic) *sentence realization*, i.e., that part of NLG which deals with mapping of a formal (more or less abstract) sentence plan onto a chain of inflected words; cf., among others, (Langkilde and Knight, 1998; Oh and Rudnicky, 2000; Bangalore and Rambow, 2000; Wan et al., 2009). The advantage of stochastic sentence realization over traditional rule-based realization is mainly threefold: (i) it is more robust, (ii) it usually has a significantly larger coverage; (iii) it is *per se* language- and domain-independent. Its disadvantage is that it requires at least syntactically annotated corpora of significant size (Bangalore et al., 2001). Given the aspiration of NLG to start from numeric time series or conceptual or semantic structures, syntactic annotation even does not suffice: the corpora must also be at least semantically annotated. Up to date, deep stochastic sentence realization was hampered by the lack of multiple-level annotated corpora. As a consequence, available stochastic sentence generators either take syntactic structures as input (and avoid thus the need for multiple-level annotation) (Bangalore and Rambow, 2000; Langkilde-Geary, 2002; Filippova and Strube, 2008), or draw upon hybrid models that imply a symbolic submodule which derives the syntactic representation that is then used by the stochastic submodule (Knight and Hatzivassiloglou, 1995; Langkilde and Knight, 1998).

The increasing availability of multilevel annotated corpora, such as the corpora of the shared task of the Conference on Computational Natural Language Learning (CoNLL), opens new perspectives with respect to deep stochastic sentence generation—although the fact that these corpora have not been annotated with the needs of generation in mind, may require additional adjustments, as has been, in fact, in the case of our work.

Proceedings of the 23rd International Conference on Computational Linguistics (Coling 2010), pages 98–106,
Beijing, August 2010

In this paper, we present a Support Vector Machine (SVM)-based multilingual dependency-oriented stochastic deep sentence realizer that uses multilingual corpora of the CoNLL '09 shared task (Hajič, 2009) for training. The sentences of these corpora are annotated with shallow semantic structures, dependency trees, and lemmata; for some of the languages involved, they also contain morphological feature annotations. The multilevel annotation allows us to take into account all levels of representation needed for linguistic generation and to model the projection between pairs of adjacent levels by separate decoders, which, in its turn, facilitates the coverage of such critical generation tasks as sentence planning, linearization, and morphologization. The presented realizer is, in principle, language-independent in that it is trainable on any multilevel annotated corpus. In this paper, we discuss its performance for Chinese, English, German, and Spanish.

The remainder of the paper is structured as follows. In Section 2, we discuss how the shallow semantic annotation in the CoNLL '09 shared task corpora should be completed in order to be suitable for generation. Section 3 presents the training setup of our realizer. Section 4 shows the individual stages of sentence realization: from the semantic structure to the syntactic structure, from the syntactic structure to the linearized structure and from the linearized structure to a chain of inflected word forms (if applicable for the language in question). Section 5 outlines the experimental set up for the evaluation of our realizer and discusses the results of this evaluation. In Section 6, finally, some conclusions with respect to the characteristics of our realizer and its place in the research landscape are drawn.

The amount of the material which comes into play makes it impossible to describe all stages in adequate detail. However, we hope that the overview provided in what follows still suffices to fully assess our proposal.

2 Completing the Semantic Annotation

The semantic annotation of sentences in CoNLL '09 shared task corpora follows the PropBank annotation guidelines (Palmer et al., 2005). Problematic from the viewpoint of generation is that this annotation is not always a connected acyclic graph. As a consequence, in these cases no valid (connected) syntactic tree can be derived. The most frequent cases of violation of the connectivity principle are not attached adjectival modifiers, determiners, adverbs, and coordinations; sometimes, the verb is not connected with its argument(s). Therefore, prior to starting the training procedure, the semantic annotation must be completed: non-connected adjectival modifiers must be annotated as predicates with their syntactic heads as arguments, determiners must be "translated" into quantifiers, detached verbal arguments must be connected with their head, etc.

Algorithm 1 displays the algorithm that completes the semantic annotations of the corpora. Each sentence x_i of the corpus I, with $i = 1, \ldots, |I|$, is annotated with its dependency tree y_i and its shallow semantic graph s_i. The algorithm traverses y_i breath-first, and examines for each node n in y_i whether n's corresponding node in s_i is connected with the node corresponding to the parent of n. If not, the algorithm connects both by a directed labeled edge. The direction and the label of the edge are selected consulting a look up table in which default labels and the orientation of the edges between different node categories are specified.

Figure 1 shows the semantic representation of a sample English sentence obtained after the application of Algorithm 1. The solid edges are the edges available in the original annotation; the dashed edges have been introduced by the algorithm. The edge labels 'A0' and 'A1' stand for "first argument" and "second argument" (of the corresponding head), respectively, 'R-A0' for "A0 realized as a relative clause", and 'AM-MNR' for "manner modifier". As can be seen, 6 out of the total of 14 edges in the complete representation of this example have been added by Algorithm 1. We still did not finish the formal evaluation of the principal changes necessary to adapt the PropBank annotation for generation, nor the quality of our completion algorithm. However, the need of an annotation with generation in mind is obvious.

Algorithm 1: Complete semantic graph

```
//$s_i$ is a semantic graph and $y_i$ a dependency tree
// $s_i = \langle N_{s_i}, L_{s_i}, E_{s_i} \rangle$, where $N_{s_i}$ is the set of nodes
// $L_{s_i}$ the set of edge labels
// $E_{s_i} \subseteq N_s \times N_s \times L_s$ is the set of edges
for $i \leftarrow 1$ to $|I|$ // iteration over the training examples
  let $r_y \in y_i$ be the root node of the dependency tree
  // initialization of the queue
  $nodeQueue \leftarrow$ children($r_y$)
  while $nodeQueue \neq \emptyset$ do
    $n_y \leftarrow$ removeFirst(nodeQueue)
    // breath first: add nodes at the end of the queue
    nodeQueue $\leftarrow$ nodeQueue $\cup$ children($n_y$)
    $n_{y_s} \leftarrow$ sem($n_y$); $p_{y_s} \leftarrow$ sem(parent($n_y$))
    //get the semantic equivalents of $n_y$ and of its parent
    if not exists path($n_{y_s}, p_{y_s}$) then
      $l \leftarrow$ label($n_y$,parent($n_y$))
      $l_s \leftarrow$ look-up-sem-label($n_{y_s}, p_{y_s}, l$)
      if look-up-sem-direction($n_{y_s}, p_{y_s}, l_s$) = "$\rightarrow$" then
        // add the semantic edge
        $E_s \leftarrow E_s \cup (p_{y_s}, n_{y_s}, l_s)$
      else // direction of the edge "$\leftarrow$"
        // add the semantic edge
        $E_s \leftarrow E_s \cup (n_{y_s}, p_{y_s}, l_s)$
```

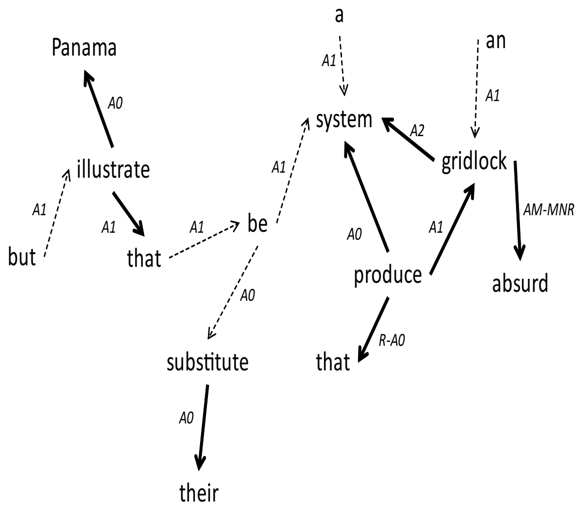

Figure 1: Semantic representation of the sentence *But Panama illustrates that their substitute is a system that produces an absurd gridlock.* after completion

3 Realizer Training Setup

Figure 2 shows the training setup of our realizer. For each level of annotation, an SVM feature extractor and for each pair of adjacent levels of annotation, an SVM decoder is defined. The Sem-Synt decoder constructs from a semantic graph the corresponding dependency tree. The Synt-Linearization decoder derives from a dependency tree a chain of lemmata, i.e., determines the word order within the sentence. The Linearization-Morph decoder generates the inflected word form for each lemma in the chain. Both the feature extractors and the decoders are language-independent, which makes the realizer applicable to any language for which multilevel-annotated corpora are available.

To compute the score of the alternative realizations by each decoder, we apply MIRA (Margin Infused Relaxed Algorithm) to the features provided by the feature extractors. MIRA is one of the most successful large-margin training techniques for structured data (Crammer et al., 2006). It has been used, e.g., for dependency parsing, semantic role labelling, chunking and tagging. Since we have similar feature sets (of comparable size) as those for which MIRA has proven to work well, we assume that it will also perform well for sentence realization. Unfortunately, due to the lack of space, we cannot present here the instantiation of MIRA for all stages of our model. For illustration, Algorithm 2 outlines it for morphological realization.

The morphologic realization uses the minimal string edit distance (Levenshtein, 1966) to map lemmata to word forms. As input to the MIRA-classifier, we use the lemmata of a sentence, its dependency tree and the already ordered sentence. The characters of the input strings are reversed since most of the changes occur at the end of the words and the string edit scripts work relatively to the beginning of the string. For example, to calculate the minimal string edit distance between the lemma *go* and the form *goes*, both are first reversed by the function **compute-edit-dist** and then the minimal string edit script between *og* and *seog* is computed. The resulting script is *Ie0Is0.* It translates into the operations 'insert *e* at the position 0 of the input string' and 'insert *s* at the position 0'.

Before MIRA starts, we compute all minimal edit distance scripts to be used as classes of MIRA. Only scripts that occur more often than twice are used. The number of the resulting edit scripts is language-dependent; e.g., we get about

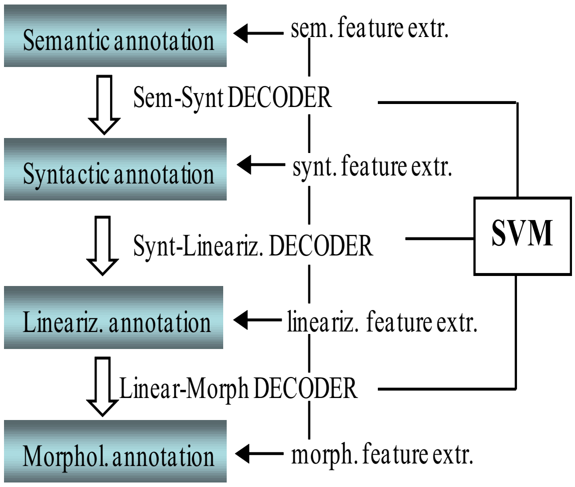

Figure 2: Realizer training scenario setup

Algorithm 2: Morphological realization training with MIRA

```
// $y_i, l_i$; $y_i$ is a dependency tree, $l_i$ lemmatized sentence
script-list ← {} //initialize the script-list
**for** $i \leftarrow 1$ **to** $|I|$ // iteration over the training examples
  **for** $l \leftarrow 1$ to $|l_i|$ **do**//// iteration over the lemmata of $l_i$
    lemma$_l \leftarrow$ *lower-case* ($l_i$,$l$)
    //ensure that all lemmata start with a lower case letter
    script ← *compute-edit-dist-script*(lemma$_l$, *form*($l_i$,$l$))
    **if** script $\notin$ script-list
      script-list ← script-list $\cup$ { script }
**for** $k \leftarrow 1$ **to** $E$ // $E$ = number of traininig epochs
  **for** $i \leftarrow 1$ **to** $|I|$ // iteration over the training examples
    **for** $l \leftarrow 1$ to $|l_i|$ **do**
      script$_p \leftarrow$ *predict-script*($l_i$,$y_i$,$l$)
      script$_g \leftarrow$ *edit-dist-script*(lemma$_l$, form($l_i$,$l$))
      **if** script$_p \neq$ script$_g$ **then**
      // update the weight vector $v$ and the vector $w$, which
      // averages over all collected weight vectors acc.
      // to diff. of the predicted and gold feature vector
      **update** $w, v$ according to $\Delta(\phi(\text{script}_p), \phi(\text{script}_g))$
      //with $\phi(\text{script}_p)$, $\phi(\text{script}_g)$ as feature vectors of
      //script$_p$ and script$_g$, respectively
```

1500 scripts for English and 2500 for German.

The training algorithms typically perform 6 iterations (*epochs*) over the training examples. For each training example, a minimal edit script is selected. If this script is different from the gold script, the features of the gold script are calculated and the weight vector of the SVM is adjusted according to the difference between the predicted vector and the *gold feature vector*. The classification task consists then in finding the classification script that maps the lemma to the correct word form. For this purpose, the classifier scores each of the minimal edit scripts according to the input, choosing the one with the highest score.

4 Sentence Generation

Sentence generation that starts from a given semantic structure as input consists in the application of the previously trained SVM decoders in sequence in order to realize the following sequence of mappings:

SemStr → *SyntStr* → *LinearStr* → *Surface*

4.1 Semantic Generation

Algorithm 3 shows the algorithm for semantic generation, i.e., the derivation of a dependency tree from a semantic structure. It is a beam search that creates a maximum spanning tree. In the first step, a seed tree consisting of one edge is built. In each of the subsequent steps, this tree is extended by one node. For the decision, which node is to be attached next and to which node, we consider the highest scoring options. This procedure works well since nodes that are close in the semantic structure are usually close in the syntactic tree as well. Therefore subtrees that contain those nodes are considered first.

Unlike the traditional n-gram based stochastic realizers such as (Langkilde and Knight, 1998), we use for the score calculation structured features composed of the following elements: (i) the lemmata, (ii) the **dist**ance between the starting node s and the target node t, (iii) the **dir**ection of the path (if the path has a direction), (iv) the sorted **bag** of in-going edges labels without repitition, (v) the **path** of edge labels between source and target node.

The composed structured features are:

- label+dist(s, t)+dir
- label+dist(s, t)+lemma$_s$+dir
- label+dist(s, t)+lemma$_t$+dir
- label+dist(s, t)+lemma$_s$+lemma$_t$+dir
- label+dist(s, t)+bag$_s$+dir
- label+dist(s, t)+bag$_t$+dir
- label+path(s, t)+dir

#	**word-pairs(w$_1$,w$_2$)**	#	**n-grams**
1	label$_{w1}$+label$_{w2}$	13	PoS$_1$+PoS$_2$+PoS$_3$
2	label$_{w1}$+lemma$_1$	14	PoS$_1$+PoS$_2$+PoS$_3$+dist
3	label$_{w1}$+lemma$_2$	15	lemma$_1$+lemma$_2$+lemma$_3$
4	label$_{w2}$+lemma$_1$	16	lemma$_1$+lemma$_2$+lemma$_3$+dist
5	label$_{w2}$+lemma$_2$	17	lemma$_1$+lemma$_3$+head(w1,w2,w3)
6	PoS$_1$+PoS$_2$	18	lemma$_1$+lemma$_3$+head(w1,w2,w3)+dist
7	PoS$_1$+PoS$_2$+head(w$_1$,w$_2$)	19	label$_1$+label$_2$+label$_3$+head(w1,w2,w3)
8	label$_{w1}$+label$_{w2}$+PoS$_1$+head(w$_1$,w$_2$)	20	label$_1$+label$_2$+label$_3$+head(w1,w2,w3)+dist
9	label$_{w1}$+label$_{w2}$+PoS$_2$+head(w$_1$,w$_2$)	21	label$_1$+label$_2$+label$_3$+lemma$_1$+PoS$_2$+head(w1,w2,w3)
10	label$_{w1}$+label$_{w2}$+PoS$_1$+PoS$_2$+head(w$_1$,w$_2$)	22	label$_1$+label$_2$+label$_3$+lemma$_1$+PoS$_2$+head(w1,w2,w3)+dist
11	label$_{w1}$+label$_{w2}$+PoS$_1$+#children$_2$+head(w$_1$,w$_2$)	23	label$_1$+label$_2$+label$_3$+lemma$_2$+PoS$_1$+head(w1,w2,w3)
12	label$_{w1}$+label$_{w2}$+PoS$_2$+#children$_1$+head(w$_1$,w$_2$)	24	label$_1$+label$_2$+label$_3$+lemma$_2$+PoS$_1$+head(w1,w2,w3)+dist
#	**global features for constituents**		
25	**if** $\vert$constituent$\vert$ > 1 **then** label$_{1st}$+label$_{last}$+label$_{last-1}$+PoS$_{first}$+PoS$_{last}$+PoS$_{head}$		
26	**if** $\vert$constituent$\vert$ > 2 **then** label$_{1st}$+label$_{2d}$+label$_{3d}$+PoS$_{last}$+PoS$_{last-1}$+PoS$_{head}$+contains-?		
27	**if** $\vert$constituent$\vert$ > 2 **then** label$_{1st}$+label$_{2d}$+label$_{3d}$+PoS$_{last}$+PoS$_{last-1}$+lemma$_{head}$+contains-?		
28	**if** $\vert$constituent$\vert$ > 3 **then** PoS$_{1st}$+PoS$_{2d}$+PoS$_{3d}$+PoS$_{4th}$+PoS$_{last}$+label$_{head}$+contains-?+pos-head		
29	**if** $\vert$constituent$\vert$ > 3 **then** PoS$_{last}$+PoS$_{last-1}$+PoS$_{last-2}$+PoS$_{last-3}$+PoS$_{first}$+label$_{head}$+contains-?+pos-head		
30	PoS$_{first}$+PoS$_{last}$+lemma$_{first}$+lemma$_{last}$+lemma$_{head}$+contains-?+pos-head		

Table 1: Feature schemas used for linearization (*label$_w$* is the label of the in-going edge to a word w in the dependency tree; *lemma$_w$* is the lemma of w, and *PoS$_w$* is the part-of-speech tag of w; *head*(w$_1$,w$_2$, ...) is a function which is 1 if w_1 is the head, 2 if w_2 is the head, etc. and else 0; *dist* is the position within the constituent; *contains-?* is a boolean value which is true if the sentence contains a question mark and false otherwise; *pos-head* is the position of the head in the constituent)

4.2 Dependency Tree Linearization

Since we use unordered dependency trees as syntactic structures, our realizer has to find the optimal linear order for the lexemes of each dependency tree. Algorithm 4 shows our linearization algorithm. To order the dependency tree, we use a one classifier-approach for all languages—in contrast to, e.g., Filippova and Strube (2009), who use a two-classifier approach for German.[1]

The algorithm is again a beam search. It starts with an elementary list for each node of the dependency tree. Each elementary list is first extended by the children of the node in the list; then, the lists are extended stepwise by the children of the newly added nodes. If the number of lists during this procedure exceeds the threshold of 1000, the lists are sorted in accordance with their score, and the first 1000 are kept. The remaining lists are removed. Afterwards, the score of each list is adjusted according to a global score function which takes into account complex features such as the first word of a consitutent, last word, the head, and the edge label to the head (cf. Table 1 for the list of the features). Finally, the nodes of the dependency tree are ordered with respect to the highest ranked lists.

Only in a very rare case, the threshold of the beam search is exceeded. Even with a rich feature set, the procedure is very fast. The linearization takes about 3 milliseconds in average per dependency tree on a computer with a 2.8 Ghz CPU.

4.3 Morphological Realization

The morphological realization algorithm selects the edit script in accordance with the highest score for each lemma of a sentence obtained during training (see Algorithm 2 above) and applies then the scripts to obtain the word forms; cf. Algorithm 5.

Table 2 lists the feature schemas used for morphological realization.

5 Experiments

To evaluate the performance of our realizer, we carried out experiments on deep generation of Chinese, English, German and Spanish, starting from CoNLL '09 shared task corpora. The size of the test sets is listed in Table 3.[2]

[1] We decided to test at this stage of our work a uniform technology for all languages, even if the idiosyncrasies of some languages may be handled better by specific solutions.

[2] As in (Langkilde-Geary, 2002) and (Ringger et al., 2004), we used Section 23 of the WSJ corpus as test set for English.

Algorithm 3: Semantic generation

```
//$s_i$, $y$ semantic graph and its dependency tree
for i ← 1 to $|I|$ // iteration over the training examples
  // build an initial tree
  for all $n_1 \in s_i$ do
  trees ← {} // initialize the constructed trees list
    for all $n_2 \in s_i$ do
      if $n_1 \neq n_2$ then
        for all $l \in$ dependency-labels do
          trees = trees ∪ {(synt($n_1$),synt($n_2$),$l$)}
  trees ← sort-trees-descending-to-score(trees)
  trees ← look-forward(1000,sublist(trees,20))
  //assess at most 1000 edges of the 20 best trees
  tree ← get-best-tree-due-to-score(trees)
  (s,t,l) ← first-added-edge(tree)
  // create the best tree
  best-tree ← (s,t,l)
  // compute the nodes that still need to be attached
  rest ← nodes($s_i$) - {$s, t$}
  while rest $\neq \emptyset$ do
    trees ← look-forward(1000,best-tree,rest)
    tree ← get-best-tree-due-to-score(trees)
    (s,t,l) ← first-added-edge(tree)
    best-tree ← best-tree ∪ { (s,t,l) }
    if (root(s,best-tree)) then rest ← rest - {$s$}
    else rest ← rest - {$t$}
```

The performance of both the isolated stages and the realizer as a whole has been assessed.

5.1 Evaluation Metrics

In order to measure the correctness of the semantics to syntax mapping, we use the unlabeled and labeled attachment score as it commonly used in dependency parsing. The labeled attachment score (LAS) is the proportion of tokens that are assigned both the correct head and the correct edge label. The unlabeled attachment score (ULA) is the proportion of correct tokens that are assigned the correct head.

To assess the quality of linearization, we use three different evaluation metrics. The first metric is the per-phrase/per-clause accuracy (*acc snt.*), which facilitates the automatic evaluation of results:

$$acc = \frac{correct\ constituents}{all\ constituents}$$

As second evaluation metric, we use a metric related to the edit distance:

$$di = 1 - \frac{m}{total\ number\ of\ words}$$

(with m as the minimum number of deletions combined with insertions to obtain the correct order (Ringger et al., 2004)).

Algorithm 4: Dependency tree linearization

```
//$y_i$ a dependency tree
for i ← 1 to $|I|$ // iteration over the training examples
  // iterate over all nodes of the dependency tree $y_i$
  for n ← 1 to $|y_i|$ do
    $subtree_n$ ← children(n) ∪ {n}
    ordered-lists$_n$ ← {} // initialize
    for all m ∈ $subtree_n$ do
      beam ← {}
      for all $l$ ∈ ordered-lists do
        beam ← beam  ∪ { append(clone($l$),m)}
      for all $l$ ∈ ordered-lists do
        score($l$) ← compute-score-for-word-list($l$)
      sort-lists-descending-to-score(beam,score)
      if | beam | > beam-size then
        beam ← sublist(0,1000,beam)
      ordered-lists$_n$ ← beam
    score$_g$($l$) ← score($l$) + compute-global-score($l$)
    sort-lists-descending-in-score(beam,score$_g$)
```

Algorithm 5: Morphological realization

```
// $y_i$ a dependency tree, and $l_i$ an ordered list of lemmata
for $l$ ← 1 to $|l_i|$ do
  script$_p$ ← predict-script($l_i$,$y_i$,$l$)
  form$_l$ ← apply-edit-dist-script(lemma$_l$, script$_p$)
```

To be able to compare our results with (He et al., 2009) and (Ringger et al., 2004), we use the BLEU score as a third metric.

For the asessment of the quality of the word form generation, we use the accuracy score. The accuracy is the ratio between correctly generated word forms and the entire set of generated word forms.

For the evaluation of the sentence realizer as a whole, we use the BLEU metric.

5.2 Experimental Results

Table 4 displays the results obtained for the isolated stages of sentence realization and of the realization as a whole, with reference to a baseline and to some state-of-the-art works. The baseline is the deep sentence realization over all stages starting from the original semantic annotation in the CoNLL '09 shared task corpora.

Note, that our results are not fully comparable with (He et al., 2009; Filippova and Strube, 2009) and (Ringger et al., 2004), respectively, since the data are different. Furthermore, Filippova and Strube (2009) linearize only English sentences

#	features
1	es+lemma
2	es+lemma+m.feats
3	es+lemma+m.feats+POS
4	es+lemma+m.feats+POS+position
5	es+lemma+(lemma+1)+m.feats
6	es+lemma+(lemma+1)+POS
7	es+lemma+(m.feats-1)+(POS-1)
8	es+lemma+(m.feats-1)+(POS-1)+position
9	es+m.feats+(m.feats-1)
10	es+m.feats+(m.feats+1)
11	es+lemma+(m.feats-1)
12	es+m.feats+(m.feats-1)+(m.feats-2)
13	es+m.feats+POS
14	es+m.feats+(m.feats+1)
15	es+m.feats+(m.feats+1)+lemma
16	es+m.feats
17	es+e0+e1+m.feats
18	es+e0+e1+e2+m.feats
19	es+e0+e1+e2+e3+m.feats
20	es+e0+e1+e2+e3+e4+m.feats
21	es+e0+m.feats

Table 2: Feature schemas used for morphological realization

Chinese	English	German	Spanish
2556	2400	2000	1725

Table 3: The number of sentences in the test sets used in the experiments

that do not contain phrases that exceed 20,000 linearization options—which means that they filter out about 1% of the phrases.

For Spanish, to the best of our knowledge, no linearization experiments have been carried out so far. Therefore, we cannot contrast our results with any reference work.

As far as morphologization is concerned, the performance achieved by our realizer for English is somewhat lower than in (Minnen et al., 2001) (97.8% vs. 99.8% of accuracy). Note, however, that Minnen et al. describe a combined analyzer-generator, in which the generator is directly derived from the analyzer, which makes both approaches not directly comparable.

5.3 Discussion

The overall performance of our SVM-based deep sentence generator ranges between 0.611 (for German) and 0.688 (for Chinese) of the BLEU score. HALogen's (Langkilde-Geary, 2002) scores range between 0.514 and 0.924, depending on the completeness of the input. The figures are not directly comparable since HALogen takes as input syntactic structures. However, it gives us an idea where our generator is situated.

Traditional linearization approaches are rule-based; cf., e.g., (Bröker, 1998; Gerdes and Kahane, 2001; Duchier and Debusmann, 2001), and (Bohnet, 2004). More recently, statistic language models have been used to derive word order, cf. (Ringger et al., 2004; Wan et al., 2009) and (Filippova and Strube, 2009). Because of its partially free order, which is more difficult to handle than fixed word order, German has often been worked with in the context of linearization. Filippova and Strube (2009) adapted their linearization model originally developed for German to English. They use two classifiers to determine the word order in a sentence. The first classifier uses a trigram LM to order words within constituents, and the second (which is a maximum entropy classifier) determines the order of constituents that depend on a finite verb. For English, we achieve with our SVM-based classifier a better performance. As mentioned above, for German, Filippova and Strube (2009)'s two classifier approach pays off because it allows them to handle non-projective structures for the *Vorfeld* within the field model. It is certainly appropriate to optimize the performance of the realizer for the languages covered in a specific application. However, our goal has been so far different: to offer an off-the-shelf language-independent solution.

The linearization error analysis, first of all of German and Spanish, reveals that the annotation of coordinations in corpora of these languages as 'X ← *and/or/...* → Y' is a source of errors. The "linear" annotation used in the PropBank ('X → *and/or/...* → Y') appears to facilitate higher quality linearization. A preprocessing stage for automatic conversion of the annotation of coordinations in the corpora would have certainly contributed to a higher quality. We refrained from doing this because we did not want to distort the figures.

The morphologization error analysis indicates a number of error sources that we will address in the process of the improvement of the model. Among those sources are: quotes at the beginning of a sentence, acronyms, specific cases of starting capital letters of proper nouns (for English and Spanish), etc.

	Chinese	English	German	Spanish
Semantics-Syntax (ULA/LAS)	95.71/86.29	94.77/89.76	95.46/82.99	98.39/93.00
Syntax-Topology (di/acc)	0.88/64.74	0.91/74.96	0.82/50.5	0.83/52.77
Syntax-Topology (BLEU)	0.85	0.894	0.735	0.78
Topology-Morphology (accuracy=correct words/all words)	–	97.8	97.49	98.48
All stages (BLEU)	0.688	0.659	0.611	0.68
Baseline (BLEU)	0.12	0.18	0.11	0.14
Syntax-Topology (He et al., 2009) (di/acc)	0.89/–	–	–	–
Syntax-Topology (He et al., 2009) (BLEU)	0.887	–	–	–
Syntax-Topology (Filippova and Strube, 2009) (di/acc)	–	0.88/67	0.87/61	–
Syntax-Topology (Ringger et al., 2004) (BLEU)	–	0.836	–	–

Table 4: Quality figures for the isolated stages of deep sentence realization and the complete process.

As far as the contrastive evaluation of the quality of our morphologization stage is concerned, it is hampered by the fact that for the traditional manually crafted morphological generators, it is difficult to find thorough quantitative evaluations, and stochastic morphological generators are rare.

As already repeatedly pointed out above, so far we intentionally refrained from optimizing the individual realization stages for specific languages. Therefore, there is still quite a lot of room for improvement of our realizer when one concentrates on a selected set of languages.

6 Conclusions

We presented an SVM-based stochastic deep multilingual sentence generator that is inspired by the state-of-the-art research in semantic parsing. It uses similar techniques and relies on the same resources. This shows that there is a potential for stochastic sentence realization to catch up with the level of progress recently achieved in parsing technologies.

The generator exploits recently available multilevel-annotated corpora for training. While the availability of such corpora is a condition for deep sentence realization that starts, as is usually the case, from semantic (predicate-argument) structures, we discovered that current annotation schemata do not always favor generation such that additional preprocessing is necessary. This is not surprising since stochastic generation is a very young field. An initiative of the generation community would be appropriate to influence future multilevel annotation campaigns or to feed back the enriched annotations to the "official" resources.[3]

The most prominent features of our generator are that it is *per se* multilingual, it achieves an extremely broad coverage, and it starts from abstract semantic structures. The last feature allows us to cover a number of critical generation issues: sentence planning, linearization and morphological generation. The separation of the semantic, syntactic, linearization and morphological levels of annotation and their modular processing by separate SVM decoders also facilitates a subsequent integration of other generation tasks such as referring expression generation, ellipsis generation, and aggregation. As a matter of fact, this generator instantiates the Reference Architecture for Generation Systems (Mellish et al., 2006) for linguistic generation.

A more practical advantage of the presented deep stochastic sentence generator (as, in principle, of all stochastic generators) is that, if trained on a representative corpus, it is domain-independent. As rightly pointed out by Belz (2008), traditional wide coverage realizers such as KPML (Bateman et al., 2005), FUF/SURGE (Elhadad and Robin, 1996) and RealPro (Lavoie and Rambow, 1997), which were also intended as off-the-shelf plug-in realizers still tend to require a considerable amount of work for integration and fine-tuning of the grammatical and lexical resources. Deep stochastic sentence realizers have the potential to become real off-the-shelf modules. Our realizer is freely available for download at `http://www.recerca.upf.edu/taln`.

[3] We are currently working on a generation-oriented multilevel annotation of corpora for a number of languages. The corpora will be made available to the community.

Acknowledgments

Many thanks to the three anonymous reviewers for their very valuable comments and suggestions.

References

Bangalore, S. and O. Rambow. 2000. Exploiting a Probabilistic Hierarchical Model for Generation. In *Proceedings of COLING '00*, pages 42–48.

Bangalore, S., J. Chen, and O. Rambow. 2001. Impact of Quality and Quantity of Corpora on Stochastic Generation. In *Proceedings of the EMNLP Conference*, pages 159–166.

Bateman, J.A., I. Kruijff-Korbayová, and G.-J. Kruijff. 2005. Multilingual Resource Sharing Across Both Related and Unrelated Languages: An Implemented, Open-Source Framework for Practical Natural Language Generation. *Research on Language and Computation*, 15:1–29.

Belz, A. 2008. Automatic generation of weather forecast texts using comprehensive probabilistic generation-space models. *Natural Language Engineering*, 14(4):431–455.

Bohnet, B. 2004. A graph grammar approach to map between dependency trees and topological models. In *Proceedings of the IJCNLP*, pages 636–645.

Bröker, N. 1998. Separating Surface Order and Syntactic Relations in a Dependency Grammar. In *Proceedings of the COLING/ACL '98*.

Crammer, K., O. Dekel, S. Shalev-Shwartz, and Y. Singer. 2006. Online Passive-Aggressive Algorithms. *Journal of Machine Learning Research*, 7:551–585.

Duchier, D. and R. Debusmann. 2001. Topological dependency trees: A constraint-based account of linear precedence. In *Proceedings of the ACL*.

Elhadad, M. and J. Robin. 1996. An overview of SURGE: A reusable comprehensive syntactic realization component. Technical Report TR 96-03, Department of Mathematics and Computer Science, Ben Gurion University.

Filippova, K. and M. Strube. 2008. Sentence fusion via dependency graph compression. In *Proceedings of the EMNLP Conference*.

Filippova, K. and M. Strube. 2009. Tree linearization in English: Improving language model based approaches. In *Proceedings of the NAACL '09 and HLT, Short Papers*, pages 225–228.

Gerdes, K. and S. Kahane. 2001. Word order in German: A formal dependency grammar using a topological hierarchy. In *Proceedings of the ACL*.

Hajič, J. et al. 2009. The CoNLL-2009 Shared Task: Syntactic and Semantic Dependencies in Multiple Languages. In *Proceedings of the CoNLL*.

He, W., H. Wang, Y. Guo, and T. Liu. 2009. Dependency based chinese sentence realization. In *Proceedings of the ACL and of the IJCNLP of the AFNLP*, pages 809–816.

Knight, K. and V. Hatzivassiloglou. 1995. Two-level, many paths generation. In *Proceedings of the ACL*.

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

Langkilde-Geary, I. 2002. An empirical verification of coverage and correctness for a general-purpose sentence generator. In *Proceedings of the Second INLG Conference*, pages 17–28.

Lavoie, B. and O. Rambow. 1997. A fast and portable realizer for text generation systems. In *Proceedings of the 5th Conference on ANLP*.

Levenshtein, V.I. 1966. Binary codes capable of correcting deletions, insertions, and reversals. *Soviet Physics*, 10:707–710.

Mellish, C., D. Scott, L. Cahill, D. Paiva, R. Evans, and M. Reape. 2006. A reference architecture for natural language generation systems. *Natural Language Engineering*, 12(1):1–34.

Minnen, G., J. Carroll, and D. Pearce. 2001. Applied morphological processing for English. *Natural Language Engineering*, 7(3):207–223.

Oh, A.H. and A.I. Rudnicky. 2000. Stochastic language generation for spoken dialogue systems. In *Proceedings of the ANL/NAACL Workshop on Conversational Systems*, pages 27–32.

Palmer, M., D. Gildea, and P. Kingsbury. 2005. The proposition bank: An annotated corpus of semantic roles. *Computational Linguistics*, 31(1):71–105.

Ringger, E., M. Gamon, R.C. Moore, D. Rojas, M. Smets, and S. Corston-Oliver. 2004. Linguistically informed statistical models of constituent structure for ordering in sentence realization. In *Proceedings of COLING*, pages 673–679.

Wan, S., M. Dras, Dale R., and C. Paris. 2009. Improving Grammaticality in Statistical Sentence Generation: Introducing a Dependency Spanning Tree Algorithm with an Argument Satisfaction Model. In *Proceedings of the EACL '09*, pages 852–860.