Sentence Generation as Planning with Probabilistic LTAG

Daniel Bauer
Department of Computer Science
Columbia University
1214 Amsterdam Ave.
New York, NY 10027, USA
bauer@cs.columbia.edu

Alexander Koller
Cluster of Excellence
Saarland University
Campus C7 4
66041 Saarbrücken, Germany
koller@mmci.uni-saarland.de

Abstract

We present PCRISP, a sentence generation system for probabilistic TAG grammars which performs sentence planning and surface realization in an integrated fashion, in the style of the SPUD system. PCRISP operates by converting the generation problem into a metric planning problem and solving it using an off-the-shelf planner. We evaluate PCRISP on the WSJ corpus and identify trade-offs between coverage, efficiency, and accuracy.

1 Introduction

Many sentence generation systems are organized in a pipeline architecture, in which the input semantic representation is first enriched, e.g. with referring expressions, by a *sentence planner* and only then transformed into natural language strings by a *surface realizer* (Reiter and Dale, 2000). An alternative approach is *integrated* sentence generation, in which both steps are performed by the same algorithm, as in the SPUD system (Stone et al., 2003). An integrated algorithm can sometimes generate better and more succinct sentences (Stone and Webber, 1998). SPUD itself gives up some of this advantage by using a greedy search heuristic for efficiency reasons. The CRISP system, a recent reimplementation of SPUD using search techniques from AI planning, achieves high efficiency without sacrificing complete search (Koller and Stone, 2007; Koller and Hoffmann, 2010).

While CRISP is efficient enough to perform well on large-scale grammars (Koller and Hoffmann, 2010), such grammars tend to offer many different ways to express the same semantic representation. This makes it necessary for the generation system to be able to compute not just grammatical sentences, but to identify which of these sentences are *good*. This problem is exacerbated when using treebank-derived grammars, which tend to underspecify the actual constraints on grammaticality and instead rely on statistical information learned from the treebank. Indeed, there have been a number of systems for statistical generation, which can exploit such information to rank sentences appropriately (Langkilde and Knight, 1998; White and Baldridge, 2003; Belz, 2008). However, to our knowledge, all such systems are currently restricted to performing surface realization, and must rely on separate modules to perform sentence planning.

In this paper, we bring these two strands of research together for the first time. We present the PCRISP system, which redefines the SPUD generation problem in terms of probabilistic TAG grammars (PTAG, (Resnik, 1992)) and then extends CRISP to solving the probabilistic SPUD generation problem using metric planning (Fox and Long, 2002; Hoffmann, 2003). We evaluate PCRISP on a PTAG treebank extracted from the Wall Street Journal Corpus (Chen and Shanker, 2004). The evaluation reveals a tradeoff between coverage, efficiency, and accuracy which we think are worth exploring further in future work.

Plan of the paper. We start by putting our research in the context of related work in Section 2 and reviewing CRISP in Section 3. We then describe PCRISP, our probabilistic extension of CRISP, in

Section 4 and evaluate it in Section 5. We conclude in Section 6.

2 Related Work

Statistical methods are popular for surface realization, but have not been used in systems that integrate sentence planning. Most statistical generation approaches follow a generate-and-select strategy, first proposed by Knight and Hatzivassiloglou (1995) in their NITROGEN system. Such systems generate a set of candidate sentences using a (possibly overgenerating) grammar and then select the best output sentence by applying a statistical language model. This family includes systems such as HALogen (Langkilde and Knight, 1998; Langkilde, 2000) and OpenCCG (White and Baldridge, 2003). The FERGUS system (Bangalore and Rambow, 2000) is a variant of this approach which, like PCRISP, employs TAG. It first assigns elementary trees to each entity in the input sentence plan using a statistical tree model and then computes the most likely derivation using only these trees with an n-gram model on the output sentence. An alternative to the n-gram based generate and select approach is to use a probabilistic grammar model, like PTAG, trained on automatic parses (Zhong and Stent, 2005). A related approach uses a model over local decisions of the generation system itself (Belz, 2008). Both models can either be used to discriminate a set of output candidates, or more directly to choose the next best decision locally. Our approach is similar in that it uses PTAG to find the most likely output structure. However, the previous work discussed so far addresses surface realization only. We extend this to a statistical NLG algorithm which does surface realization and sentence planning at the same time.

Our treatment of integrated sentence planning and surface realization as planning is inherited directly from CRISP (Koller and Stone, 2007). Planning has long played a role in generation, but has focused on discourse planning instead of specifically addressing sentence generation (Hovy, 1988; Appelt, 1985). The applicability of these ideas was limited at that time because efficient planning technology was not available. Recently the development of more efficient planning algorithms (Hoffmann and Nebel, 2001) spawned a renewed interest in planning for NLG. CRISP uses such algorithms to efficiently solve the sentence generation problem defined by SPUD (Stone et al., 2003). SPUD, which instead uses an incomplete greedy algorithm, is based on a TAG whose trees are augmented with semantic and pragmatic constraints. Given a communicative goal, a solution to the SPUD problem realizes this goal and simultaneously selects referring expressions. The next section explores CRISP in more detail.

3 Sentence Generation as Planning

In this section we review the original non-statistical CRISP system (Koller and Stone, 2007). Following SPUD (Stone et al., 2003), CRISP is based on a declarative description of the sentence generation problem using TAG. Given a knowledge base, a communicative goal and a grammar, we require to find a grammatical TAG derivation that is consistent with this knowledge base and satisfies a communicative goal. A number of semantic and pragmatic constraints that must be satisfied by the solution can be added, for instance to enforce generation of unambiguous referring expressions. Koller and Stone (2007) describe how to encode this problem into an AI planning problem which can be solved efficiently by off-the-shelf planners. We describe the general mechanism in the following section and then review the encoding into planning in section 3.2.

3.1 Sentence Generation in CRISP

Like SPUD, CRISP uses an LTAG in which elementary trees are assigned semantic content. Each node in a CRISP elementary tree is associated with a semantic role. Semantic content is expressed as a set of literals, encoding relations between these roles. All nodes that dominate the lexical anchor are assigned the role 'self', which intuitively corresponds to the event or individual described by this tree. Fig. 1(a) shows an example grammar of this type.

In a derivation we may only include elementary trees whose semantic content has an instantiation in the knowledge base. For each substitution and adjunction, the semantic role associated with the role of the target node is unified with the 'self' role of the child tree. For example, given the knowledge base

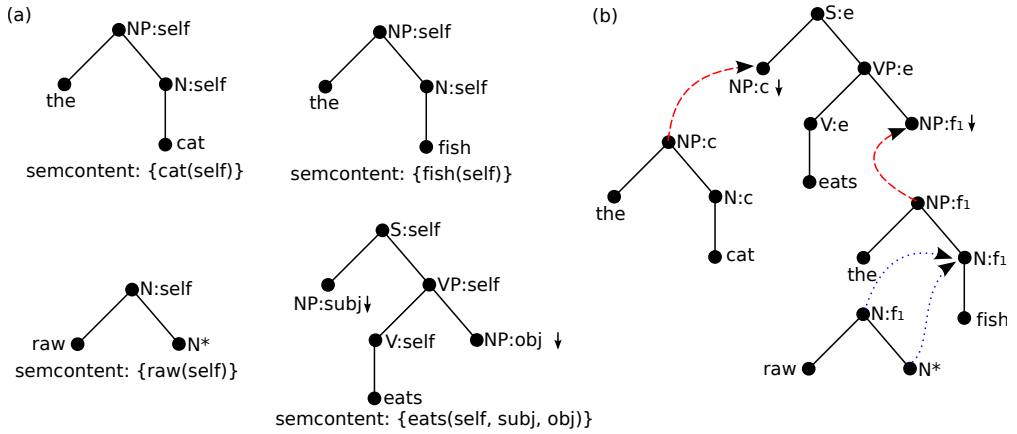

Figure 1: (a) An example grammar with semantic content. (b) A derivation for "the cat eats the raw fish".

$\{\text{cat}(c), \text{fish}(f_1), \text{raw}(f_1), \text{fish}(f_2), \text{eats}(e, c, f_1)\}$ and the grammar in 1(a), CRISP could produce the derivation in 1(b). Notice that CRISP can generate the unambiguous referring expression 'the raw fish' for f_1, to distinguish it from f_2.

3.2 CRISP Planning Domains

Before we describe CRISP's encoding of sentence generation as planning, we briefly review AI planning in general. A planning state is a conjunction of first order literals describing relations between some individuals. A planning problem consist of an initial state, a set of goal states and a set of planning operators that describe possible state transitions. A *plan* is any sequence of actions (instantiated operators) that leads from the initial state to one of the goal states. Planning problems can be solved efficiently by general purpose planning systems such as FF (Hoffmann and Nebel, 2001).

In CRISP, planning states correspond to partial TAG derivations and record open substitution and adjunction sites, semantic individuals associated with them, and parts of the communicative goal that have not yet been expressed. The initial state also encodes the knowledge base and the communicative goal. Each planning operator contributes a new elementary tree to the derivation and at the same time can satisfy part of the communicative goal, as described in the previous section. In a goal state there are no open substitution sites left and all literals in the communicative goal have been expressed.

Fig. 2 shows planning operators for part of the grammar in Fig. 1(a). The operators are simplified for lack of space, and in particular we do not show the preconditions and effects that enforce uniqueness of referring expressions; see Koller and Stone (2007) for details. The preconditions of the operators require that a suitable open substitution node (i.e. of the correct category) or internal node for adjunction exists in the partial derivation. In the operator effect, open substitution nodes are closed and new identifiers are created for each substitution node and internal node in the new tree. Given the knowledge base from above, a plan corresponding to the derivation in

subst-t28-eats-S(u, x1, x2, x3):
Precond: referent(u, x1),
subst(S, u), eats(x1, x2, x3)
Effect: ¬needtoexpr(pred-eats, x1, x2, x3),
¬subst(S, u),
subst(NP, subj), subst(NP, obj),
referent(subj, x2), referent(obj, x3)
adj(VP, u), adj(V, u), adj(S, u)

subst-t3-cat-NP(u, x1):
Precond: referent(u, x1),
subst(NP, u), cat(x1)
Effect: ¬needtoexpr(pred-cat, x1),
¬subst(NP, u),
adj(N, u), adj(NP, u)

adj-t5-raw-N(u, x1):
Precond: referent(u, x1),
adj(N, u), raw(x1)
Effect: ¬needtoexpr(pred-raw, x1),
¬adj(N, u)

Figure 2: CRISP operators for some of the elementary trees in Fig. 1.

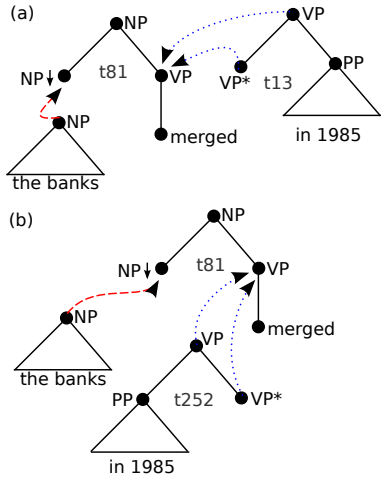

Figure 3: Two derivations with a large grammar, that satisfy the same communicative goal. Sentence (b) is dispreferred by most readers.

Fig. 1 would be subst-t28-eats-S-eats(root, e, c, f_1); subst-t3-cat-NP(subj, c); subst-t3-fish-NP(obj, f_1); adj-t5-raw-N(obj, f_1). This plan can be automatically decoded into a derivation tree for Fig. 1b.

3.3 CRISP and Large Grammars

Koller and Hoffmann (2010) report on experiments that show CRISP can generate sentences with the large-scale XTAG grammar (XTAG Research Group, 2001) quite efficiently. However, because CRISP has no notion of how "good" a generated sentence is compared to other grammatical alternatives, it will sometimes compute dispreferred sentences with large grammars. This is especially true for treebank-induced grammars, which tend to overgenerate and rely on statistical methods to rank good sentences highly. Fig. 3 illustrates this problem. Assuming a (treebank) grammar that includes trees for both right adjoining (t13) and left adjoining PPs (t252), both derivations (a) and (b) are *grammatical* derivations that satisfy the same communicative goal. However, most readers disprefer the reading in (b). Clearly, to use CRISP with such a grammar we need a method of distinguishing good derivations from bad ones.

4 Statistical Generation as Planning

We now extend CRISP to statistical generation (PCRISP). The basic idea is to add a statistical grammar model while leaving the sentence generation mechanism untouched. This way we can select the highest scoring derivation which satisfies all constraints (grammaticality, expresses the communicative goal, uses unambiguous referring expressions, etc.).

As a straightforward probability model over LTAG derivations we choose probabilistic TAG (PTAG) (Resnik, 1992). Our choice of PTAG for sentence generation is motivated by a number of attractive properties. PTAG is lexicalized and therefore does not only assign probabilities to operations in the grammar (as for example plain PCFG), but also accounts for binary dependencies between words. Unlike n-gram models however, these co-occurrences are structured according to local syntactic context as a result of TAG's extended domain of locality. The probability model describes how the syntactic arguments of a word are typically filled. Furthermore, as TAG factors recursion from the domain of dependencies, the probability for core constructions remains the same independent of additional adjunctions. We review PTAG in section 4.1.

While we leave the basic sentence generation mechanism intact, we need to modify the concrete formulation of CRISP planning operators to accommodate bilexical dependencies. Likewise, we need to take the step from classical planning to *metric* planning systems which can use the probabilities. In metric planning (Fox and Long, 2002), planning actions can modify the value of numeric variables in addition to adding and deleting logical literals from the state. The goal state specifies constraints on this variable. In the simplest case the variable can only be increased by a static cost value in each action, and the goal state contains the objective to minimize the total cost. While systems such as Metric-FF (Hoffmann, 2003) do not guarantee optimality, they do generally offer good results. We address our encoding of sentence generation with PTAG as metric planning in section 4.2.

4.1 Probabilistic TAG

PTAG (Resnik, 1992) views TAG derivations as sequences of events of three types: initial events, substitution events, and adjunction events.

The probability distribution for initial events describes how likely it is to start any derivation with a

given initial tree. It is defined over all initial trees $\alpha \in I$ with their possible lexicalizations $w \in W_\alpha$:

$$\sum_{\alpha \in I} \sum_{w \in W_\alpha} P_i(\mathit{init}(\alpha, w)) = 1$$

For substitution events, there is a probability distribution for each substitution node n of each elementary tree τ lexicalized with v, which describes how likely it is to substitute it with an initial tree α lexicalized with w.

$$\sum_{\alpha \in I} \sum_{w \in W_\alpha} P_s(\mathit{subst}(\tau, v, \alpha, w, n)) = 1$$

Similarly for each internal node there is a distribution that describes the probability to adjoin an auxiliary tree $\beta \in A$ lexicalized with w. In addition some probability mass is reserved for the event of not adjoining anything to such a node at all.

$$P_a(\mathit{noadj}(\tau, v, n)) + \sum_{\beta \in A} \sum_{w \in W_\beta} P_a(\mathit{adj}(\tau, v, \beta, w, n)) = 1.$$

PTAG assumes that all events occur independently of each other. Therefore it defines the total probability for a derivation as the product of the probability of its individual events.

4.2 PCRISP Planning Domains

Using the definition of PTAG, we now reformulate the CRISP planning operators described in section 3.2. The independence assumption in PTAG allows us to continue to model each addition of a single elementary tree to the derivation (with a certain probability score). However, while CRISP planning operators can add an elementary tree to any site of the correct category, PTAG substitution and adjunction events are binary events between lexicalized trees at a specific node. We therefore adapt the literals that record open substitution and adjunction sites in partial derivations accordingly and create one operator for each node in each possible combination of lexicalized trees. Fig. 4 shows an example planning operator for each type.

Finally, we set the cost of an operator to be its negative log probability. For example

$$\mathit{Cost}(\text{subst-t3-cat-t28-eats-n1}) = -\log P_s(\mathit{subst}(\text{t3}, \textit{'cat'}, \text{t28}, \textit{'eats'}, \text{n1}))).$$

subst-t3-cat-t28-eats-n1(u, x1):
Precond: referent(u, x1), subst(t28-eats, n1, u), cat(x1)
Effect: ¬needtoexpr(pred-cat, x1), ¬subst(t-28-eats, n1, u), adj(t3-cat, n2 u)
Cost: 4.3012

adj-t5-raw-t3-fish-n2(u, x1):
Precond: referent(u, x1), adj(t28-eats,n2, u), raw(x1)
Effect: ¬needtoexpr(pred-raw, x1), ¬adj(t-28-eats, n2, u)
Cost: 6.9076

init-t28-eats(u, x1, x2, x3):
Precond: referent(u, x1), eats(x1, x2, x3)
Effect: ¬needtoexpr(pred-eats, x1, x2, x3), subst(t-28-eats, n1,subj), subst(t28-eats, n4,obj), adj(t-28-eats, n2, u), adj(t-28-eats, n3, u)
Cost: 8.5172

noadj-t28-eats-n3(u):
Precond: adj(t-28-eats, n3,u)
Effect: ¬ adj(t-28-eats, n3, u)
Cost: 0.1054

Figure 4: Some PCRISP operators for the grammar from Fig. 1.

This way the plan which minimizes the sum of the costs of its actions corresponds to the TAG derivation with the highest probability.

4.3 Dealing with Data Sparseness

The event definition of PTAG is very-fine grained. Substitution and adjunction events depend on specific parent and child trees with specific lexicalizations and on a node in the parent tree, as illustrated in Fig. 5, B.1. When we estimate a probability model from training data, we cannot expect to observe evidence for all combinations of trees. Derivations that include such unseen events have zero probability and are therefore impossible. As we show in section 5, this gives rise to a massive data sparseness problem.

A straightforward way to deal with data sparseness is to drop all lexicalizations from event definitions, as illustrated in Fig. 5, A. Unfortunately this model no longer accounts for bilexical dependencies between words. Since our system has to add a lexicalized tree in each step, lexicalizations for this child tree should always be taken into account by the probability model, if available. Despite these drawbacks, we perform experiments with the unlexical-

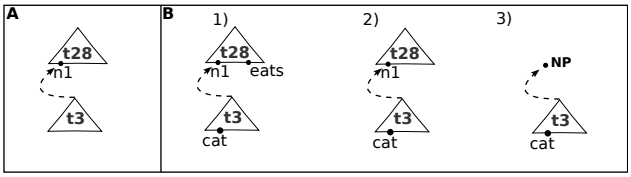

Figure 5: Illustration of the unlexicalized probability model (A) and the three back-off levels of the linear interpolation model (B).

ized model as a baseline. This allows us to investigate if purely syntactic information is sufficient to achieve high quality generation output.

An alternative model computes linear interpolation between three back-off levels. The first level is just standard PTAG (Fig. 5, B.1), for the second level the lexicalizations of the parent tree is dropped (Fig. 5, B.2), for the third level the model describes only the distribution of lexicalized child trees over each category (Fig. 5, B.3). Notice also that the third level is similar in spirit to a probabilistic version of original CRISP operators (compare Fig. 5, B.3 and Fig. 2).

5 Evaluation

We now report on some experiments with PCRISP. We are interested in the output quality and runtime behavior of different combinations of planning systems and probability models. Like CRISP, PCRISP is an integrated sentence generation system capable of generating referring expressions during surface realization. However, for lack of an appropriate evaluation dataset for full sentence generation, we only evaluate PCRISP on a realization task here. Further experiments and more details can be found in Bauer (2009).

5.1 Evaluation Data

For our experiments we use an LTAG grammar and treebank that was automatically extracted from the Wall Street Journal using the algorithm described by Chen and Shanker (2004).

This algorithm outputs a grammar that allows multiple adjunctions at the same node. For such a grammar PTAG is not a suitable probability model. Models that can deal with multiple adjunction are discussed by Nasr and Rambow (2006), but employing them would require non-trivial modifications to our encoding as a metric planning problem. We therefore preprocess the treebank by linearizing multiple adjunctions. Furthermore, we reattach prepositions to the trees they substitute in, to increase the expressiveness of the bilexical probability distribution.

We then automatically create semantic content for all lexicalized elementary trees by assigning a single semantic literal to each tree in the treebank, using the lexical anchor as the predicate symbol and variables for each substitution node and the 'self' role as arguments. We calculate the role associated with each node in each tree by assigning role names to each substitution node ('self' is assigned to the lexical anchor) and then percolating the roles up the tree, giving preference to the 'self' role.

We estimate our probability models on section 1 to 23 of the converted WSJ using maximum likelihood estimation. We use section 0 as a testing set. However, since the number of PCRISP operators grows quadratically with the grammar size, generating long sentences requires too much time to run batch experiments. We therefore restrict our evaluation to the 416 sentences in Section 0 that are shorter than 16 words.

For this testing set we automatically create semantic representation for each sentence, by instantiating the semantic content of each elementary tree used in its derivation. We use these representations as input for our system and compare the system output against the original sentence.

5.2 Generation tree accuracy

To evaluate the output quality of our statistical generation system we compare the system output O against the reference sentence R in the treebank from which the input representation was generated. We adopt the **generation tree accuracy** (GTA) measure discussed by (Bangalore et al., 2000). This measure is computed by first creating a list of all 'treelets' from the reference derivation tree D. A 'treelet' is a subtree consisting only of a node and its direct children. For each treelet we calculate the edit distance, sum the distances over all treelets and

then divide by the total length of the reference string:

$$1 - \frac{\sum_{d \in treelets(D)} editDist(O|_d, R|_d)}{length(R)},$$

where D is the reference derivation tree and $S|_d$ are the tokens associated with the nodes of treelet d in the order they appear in S (if at all). Edit distance is modified to treat insertion-deletion pairs as single movement errors. Compared to a purely string-based metric like BLEU, GTA penalizes swapped words less harshly if they can be explained by local tree movements.

5.3 Results

Table 1 presents the results of the experiment for five different generation systems. We compare three variants of PCRISP: the fully lexicalized PTAG model ("PTAG"), the fully unlexicalized model ("unlexicalized"), and a linear interpolation model ("interpolation") in which we (manually) set the weight of level 2 to 0.9 and the weight of level 3 to 0. We also list results for the non-statistical CRISP system of Koller and Stone (2007) ("CRISP") and the greedy search heuristic used by SPUD. All these systems are based on a reimplementation of the FF planner (Hoffmann and Nebel, 2001), to which we added a search heuristic that takes action costs into account for the PCRISP systems.

For each system, we determine the proportion of sentences in the test set for which the system produced an output ("success"), did not find a plan ("fail"), and exceeded the five-minute timeout ("timeout"). The column "gta" records the mean generation tree accuracy for those sentences where the system produced an output.

The table confirms that the non-statistical CRISP system has considerable trouble reconstructing the original sentences from the treebank, with a mean GTA of 0.66. This is still better than our reimplementation of SPUD, which fails to recover from early mistakes of its greedy search heuristic for every single sentence in the test set.

By contrast, the fully lexicalized PCRISP model achieves a much better mean GTA. However, this comes at the cost of a very low success rate of only 10%, reflecting a serious data sparseness problem on unseen inputs. The data sparseness problem is reduced in the unlexicalized version of PCRISP but this comes at the cost of decreased accuracy and much increased runtimes. The linear interpolation model strikes a balance between these two, by improving the success rate over the lexicalized model while sacrificing only a small amount of accuracy. This suggests that smoothing is a promising approach to balancing coverage, efficiency, and accuracy, but clearly further experimentation is needed to substantiate this.

System	gta	success	fail	timeout
SPUD	n/a	0%	100%	0%
CRISP	0.66	45%	42%	13%
PTAG	0.90	10%	88%	2%
unlexicalized	0.74	62%	16%	22%
interpolation	0.88	19%	74%	7%

Table 1: Results for the realization experiment.

6 Conclusion

We have described PCRISP, an approach to integrated sentence generation which can compute the best derivation according to a probabilistic TAG grammar. This brings two strands of research – statistical generation and integrated sentence planning and realization – together for the first time. Our generation algorithm operates by converting the generation problem into a metric planning problem and solving it with an off-the-shelf planner. An evaluation on the WSJ corpus reveals that PCRISP, like PTAG in general, is susceptible to data sparseness problems. Because the size of the planning problem is quadratic in the number of lexicalized trees in the grammar, current planning algorithms are also too slow to be used for longer sentences.

An obvious issue for future research is to apply improved smoothing techniques to deal with the data sparseness. Planning runtimes should be improved by further tweaking the exact planning problems we generate, and will benefit from any future improvements in metric planning. It is interesting to note that the extensions we made to CRISP to accommodate statistical generation here are compatible with recent work in which CRISP is applied to situated generation (Garoufi and Koller, 2010); we expect that this will be true for other future extensions to CRISP as

well. Finally, we have only evaluated PCRISP on a surface realization problem in this paper. It would be interesting to carry out an extrinsic, task-based evaluation of PCRISP that also addresses sentence planning.

Acknowledgments. We are grateful to Owen Rambow for providing us with the Chen WSJ-TAG corpus and to Malte Helmert and Silvia Richter for their help with running LAMA, another metric planner with which we experimented. We thank Konstantina Garoufi and Owen Rambow for helpful discussions, and our reviewers for their insightful comments.

References

D. Appelt. 1985. *Planning English Sentences*. Cambridge University Press, New York, NY.

S. Bangalore and O. Rambow. 2000. Exploiting a probabilistic hierarchical model for generation. In *Proceedings of ACL-2000*.

S. Bangalore, O. Rambow, and S. Whittaker. 2000. Evaluation metrics for generation. In *Proceedings of INLG-2000*.

D. Bauer. 2009. Statistical natural language generation as planning. Master's thesis, Department of Computational Linguistics, Saarland University, Saarbrücken, Germany. `http://www.coli.uni-saarland.de/~dbauer/documents/MSc_Bauer2009.pdf`.

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

J. Chen and V.K. Shanker. 2004. Automated extraction of TAGs from the Penn treebank. In Bunt H., J. Carrol, and G. Satta, editors, *New Developments in Parsing Technology*, pages 73–89. Kluwer, Norwell, MA.

M. Fox and D. Long. 2002. The third international planning competition: temporal and metric planning. In *Proceedings of the International Conference on Artificial Intelligence Planning and Scheduling (AIPS-02)*, pages 333–335.

K. Garoufi and A. Koller. 2010. Automated planning for situated natural language generation. In *Proceedings of ACL-2010*.

J. Hoffmann and B. Nebel. 2001. The FF planning system: fast plan generation through heuristic search. *Journal of Artificial Intelligence Research*, 14:253–302.

J. Hoffmann. 2003. The Metric-FF planning system: Translating "ignoring delete lists" to numeric state variables. *Journal of Artificial Intelligence Research*, 20:291–341.

E. Hovy. 1988. *Generating Natural Language Under Pragmatic Constraints*. Lawrence Erlbaum, Hillsdale, NJ.

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

A Koller and J. Hoffmann. 2010. Waking up a sleeping rabbit: On natural-language generation with FF. In *Proceedings of ICAPS-2010*.

A. Koller and M. Stone. 2007. Sentence generation as planning. In *Proceedings of ACL 2007*.

I. Langkilde and K. Knight. 1998. Generation that exploits corpus-based statistical knowledge. In *Proceedings of ACL-1998*.

I. Langkilde. 2000. Forest-based statistical sentence generation. In *Proceedings of NAACL-HLT 2000*.

A. Nasr and O. Rambow. 2006. Parsing with lexicalized probabilistic recursive transition networks. In *Finite State Methods and Natural Language Processing*, volume 4002 of *Lecture Notes in Computer Science*, pages 156–166. Springer.

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

P. Resnik. 1992. Probabilistic tree-adjoining grammar as a framework for statistical natural language processing. In *Proceedings of COLING-1992*.

M. Stone and B. Webber. 1998. Textual economy through close coupling of syntax and semantics. In *Proceedings of INLG-98*.

M. Stone, C. Doran, B. Webber, T. Bleam, and M. Palmer. 2003. Microplanning with communicative intentions: The SPUD system. *Computational Intelligence*, 19(4):311–381.

M. White and J. Baldridge. 2003. Adapting chart realization to CCG. In *Proceedings of ENLG-2003*.

XTAG Research Group. 2001. A lexicalized tree adjoining grammar for english. Technical Report IRCS-01-03, IRCS, University of Pennsylvania.

H. Zhong and A. Stent. 2005. Building surface realizers automatically from corpora. *Proceedings of UCNLG05*.