Training a Sentence Planner for Spoken Dialog: The Impact of Syntactic and Planning Features

Monica Rogati, Marilyn Walker, Owen Rambow

Carnegie Mellon University, AT&T Labs – Research, AT&T Labs – Research
rogati@cs.cmu.edu, walker,rambow@research.att.com

Abstract

The dialog manager of a spoken dialog system often performs domain dependent functions as well as general dialog tasks. It is possible to separate the domain specific knowledge from knowledge about language using techniques from natural language generation. However a natural language generator often has to be tuned for particular applications. In this work, we describe a new method for automatically training the natural language generator and examine the role that domain specific and domain independent features have on performance. We show that although the general features have the largest impact, the use of domain specific features improves performance, while still retaining the benefits of automatic domain customization through training.

1. Introduction

One of the most challenging problems for spoken dialog systems is the design of the system's utterance generation module. This challenge arises from the fact that the utterance generator needs to be sensitive to, and adapt to, many features of the dialog domain, user population and dialog context. Most dialog systems today simply output completely formed utterances using template-based generation, with templates customized anew for each application domain. In order to create re-usable and automatically customizable components, our research applies techniques from *natural language generation* which divides the generation process into three modules: (1) *text planning*; (2) *sentence planning*; and (3) *surface realization*. This paper focuses on *sentence planning*.

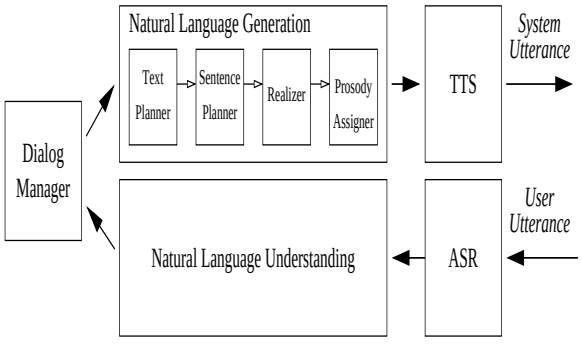

Figure 1: Architecture of a dialog system with a generator

Figure 1 shows the overall architecture of our spoken dialog system using natural language generation. A mixed-initiative spoken dialog system for travel planning must be able to generate utterances such as those in System5 in Dialog D1.

D1 System1: Welcome.... What airport would you like to fly out of?
User2: I need to go to Dallas.
System3: Flying to Dallas. What departure airport was that?
User4: from Newark on September the 1st.
System5: What time would you like to travel on September the 1st to Dallas from Newark?

The system's communicative goals for System5 are in Figure 2. In a mixed-initiative system, any combination of these goals and many others can occur in any system turn. This is because the communicative goals arise in direct response to the amount of initiative that the user takes and which task information the user takes the initiative to provide. A challenge for high quality utterance generation is that there are many ways to communicate combinations of multiple communicative goals; some examples for System5 are in Figure 3.

implicit-confirm(orig-city:NEWARK)
implicit-confirm(dest-city:DALLAS)
implicit-confirm(month:9)
implicit-confirm(day-number:1)
request(depart-time:whatever)

Figure 2: The text plan (communicative goals) for System5 in Dialog D1

Alt	Realization
0	What time would you like to travel on September the 1st to Dallas from Newark?
5	Leaving on September the 1st. What time would you like to travel from Newark to Dallas?
13	Now, what time would you like to leave? Flying to DALLAS from Newark on September the 1.

Figure 3: Some of Many Alternative Sentence Plan Realizations for the Text Plan for Utterance System5 in Dialog D1.

The job of the sentence planner is to choose linguistic resources to realize a text plan. Each text plan is an unordered set of elementary speech acts encoding the system's communicative goals for the current turn, as in Figure 2. Each speech act is represented as a type (request, implicit confirm, explicit confirm), with type-specific parameters. The sentence planner decides among alternative realizations of this text plan.

In [7], we present a new sentence planner, **SPoT**, and a methodology for *automatically training* it using human feedback. We show that **SPoT** learns to select plans whose rating on

average is only 5% worse than the top human-ranked sentence plan. However, because our approach uses both domain dependent and independent features, an immediate question is which aspects of training will carry over to a new domain.

In this paper, we train **SPoT** with different feature sets, and separate the more general, linguistic features from their domain-specific lexicalized counterparts. We then compare the performance of the different features with a HUMAN topline. We describe the architecture, training methods and results.

2. Generator Architecture

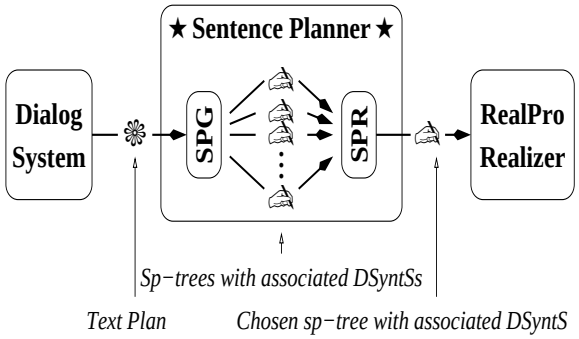

Figure 4: Architecture of **SPoT**

The architecture of **SPoT** is in Figure 4. As the figure shows, sentence planning is divided into two phases. First, the sentence-plan-generator (**SPG**) generates 12-20 possible sentence plans for the input text plan. Second, the sentence plan ranker (**SPR**) ranks the generated plans, and then selects the top-ranked plan as input to the surface realizer, RealPro [3].

A strength of our approach is the ability to use a very simple **SPG**. The basis of our **SPG** is a set of clause-combining operations that incrementally transform a list of elementary predicate-argument representations (lexico-structural representations called **DSyntS** – [4]) into a list of lexico-structural representations of single sentences, by combining them using the operations exemplified in Figure 5[5, 6]. The result of applying the operations is a **sentence plan tree** (or **sp-tree** for short), which is a binary tree with leaves labeled by the speech acts from the input text plan, and interior nodes labeled with clause-combining operations.

The complexity of most sentence planners arises from the attempt to encode constraints on the application of, and ordering of, the operations, in order to generate a single high quality sentence plan. In our approach, we do not need to encode such constraints. Rather we generate a random sample of possible sentence plans for each text plan, up to a pre-specified maximum number of sentence plans, by randomly selecting among the operations according to a probability distribution which favors preferred operations.

Figure 3 shows some of the utterances generated by our **SPG** for System5 in Dialog D1. The sp-tree for alternative 5 is in Figure 6. Node **soft-merge-general**$_3$ merges an implicit-confirmation of the destination city and the origin city. The row labelled SOFT-MERGE in Figure 5 shows the result of the soft-merge operation when Args 1 and 2 are implicit confirmations of the origin and destination cities.

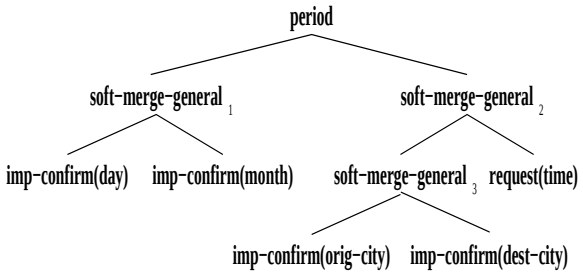

Figure 6: Alternative 5 Sentence Plan Tree

3. Training The Sentence-Plan-Ranker

The sentence-plan-ranker **SPR** takes as input a set of sentence plans generated by the **SPG** and ranks them. In order to train the **SPR** we apply the machine learning program RankBoost [2], and learn a set of rules for ranking sentence plans from a labelled set of sentence-plan training examples.

Examples and Feedback: To apply RankBoost, we use a set of example sp-trees, rate each of them, and encode them in terms of a set of features. More specifically, we start with a corpus of 100 text plans generated in context in 25 dialogs by the dialog system. We then run the **SPG**, parameterized to generate at most 20 distinct sentence plans for each text plan; this results in a corpus of 1868 sentence plans. These 1868 sentence plans, realized by RealPro, were then rated by two expert judges in the context of the original dialogs (transcribed), on a scale from 1 to 5, and the ratings averaged.[1] The ratings assigned to the sentence plans were roughly normally distributed, with a mean of 2.86 and a median of 3. Each sp-tree provided an example input to RankBoost, with each corresponding rating its feedback.

Features used by Rankboost: RankBoost requires each example to be encoded as a set of real-valued features. (Binary features are modeled with values 0 and 1.) A strength of RankBoost and similar machine learning programs is that the set of features can be very large. We use 3,291 features for training the **SPR**. These features count the number of occurrences of certain structural configurations, in order to capture declaratively decisions made by the randomized **SPG**. The features are based on feature templates and were automatically extracted (using the templates) from the set of sentence plan trees and the associated DSyntS trees that the **SPG** generated. The templates and the feature discovery process is described below (see also [1]).

For this experiment, we distinguish two classes of features: (1) **Sp-features**. These features are derived from the sp-trees and represent the way in which the rules are applied to the elementary speech acts. These features do not reflect the lexical realizations of the speech acts and are the most general. The feature names are prefixed with "SP". (2) **DSyntS-features**. These features are derived from the DSyntSs associated with the root nodes of sp-trees. They describe the overall deep-syntactic structure of the utterance, including the chosen lexemes. Therefore, they have the tendency to be domain-specific. The feature names are prefixed with "DSYNT".

We now describe the feature templates used in the discovery process. Three of these templates were used for both the sp-features and the DSyntS-features, while two were used only for the sp-features. We distinguish between *local feature templates* which record structural configurations local to a particular node

[1] Adapting **SPoT** to particular users simply requires those users to provide the training feedback.

Rule	Arg 1	Arg 2	Result
MERGE	You are leaving from Newark.	You are leaving at 5	You are leaving at 5 from Newark
MERGE-GENERAL	What time would you like to leave?	You are leaving from Newark.	What time would you like to leave from Newark?
SOFT-MERGE	You are leaving from Newark	You are going to Dallas	You are traveling from Newark to Dallas
SOFT-MERGE-GENERAL	What time would you like to leave?	You are going to Dallas.	What time would you like to fly to Dallas?
CONJUNCTION	You are leaving from Newark.	You are going to Dallas.	You are leaving from Newark and you are going to Dallas.
RELATIVE-CLAUSE	Your flight leaves at 5.	Your flight arrives at 9.	Your flight, which leaves at 5, arrives at 9.
ADJECTIVE	Your flight leaves at 5.	Your flight is nonstop.	Your nonstop flight leaves at 5.
PERIOD	You are leaving from Newark.	You are going to Dallas.	You are leaving from Newark. You are going to Dallas
RANDOM-CUEWORD	What time would you like to leave?	n/a	Now, what time would you like to leave?

Figure 5: List of clause combining operations with examples

(such as its ancestors, its daughters, etc.), and *global feature templates*, which are used only for sp-features and record properties of the entire sp-tree. There are four types of local feature templates. All local feature templates are instantiated for all nodes in an sp-tree or in a DSyntS tree (except that the LEAF feature is not instantiated in DSyntS trees); the value of the resulting feature is the number of times the described configuration is found in the sp-tree or the DSyntS tree. In all cases, we avoid features specific to particular text plans by discarding those occur fewer than 10 times. We now present the four local and the global feature templates.

Traversal features: For each node in the tree, features are generated that record the preorder traversal of the subtree rooted at that node, for all subtrees of all depths (starting with a single-node traversal which just looks at the current node, up to the maximum depth). Feature names are constructed with the prefix "tvl-", followed by the concatenated names of the nodes (starting with the current node) on the traversal path.

Sister features: These features record all consecutive sister nodes. Names are constructed with the prefix "SIS-", followed by the concatenated names of the sister nodes. As an example, consider the sp-tree shown in Figure 6. The feature SP-SIS-IMPLICIT-CONFIRM*IMPLICIT-CONFIRM describes the configuration of implicit confirms; its value is 2.

Ancestor features: For each node in the tree, these features record all the initial subpaths of the path from that node to the root. Feature names are constructed with the prefix "anc-", followed by the concatenated names of the nodes (starting with the current node). For example, the feature SP-ANC*IMPLICIT-CONFIRM-ORIG-CITY*SOFT-MERGE-GENERAL*SOFT-MERGE-GENERAL counts the number two soft-merge-general nodes dominating an implicit confirm of the origin; its value is 1 in Figure 6.

Leaf features: These features record all initial substrings of the frontier of the sp-tree (recall that its frontier consists of elementary speech acts). Names are prefixed with "leaf-", followed by the concatenated names of the frontier nodes (starting with the current node). The value is always 0 or 1. For example, the sp-tree of Figure 6 has value 1 for features LEAF-IMPLICIT-CONFIRM AND LEAF-IMPLICIT-CONFIRM*IMPLICIT-CONFIRM, representing the first two sequences of speech acts on the leaves of the tree.

Global Features: These features only apply to the sp-tree. They record, for each sp-tree and for each operation labeling a non-frontier node (i.e., rule such as CONJUNCTION or MERGE-GENERAL), (1) the minimal number of leaves (elementary speech acts) dominated by a node labeled with that rule in that tree (MIN); (2) the maximal number of leaves dominated by a node labeled with that rule (MAX); and (3) the average number of leaves dominated by a node labeled with that rule (AVG). For example, the sp-tree for in Figure 6 has value 3 for SOFT-MERGE-GENERAL-MAX, 2 for -MIN, and 2.33 for -AVG.

4. Experimental Results

We test **SPoT** using 5-fold cross-validation (jack-knifing). The evaluation metric is the human-assigned score for the variants chosen by **SPoT**. We extracted the sp- and DSyntS-features separately and we trained using 1000 rounds of boosting for each of them, as well as for the combined feature set. We evaluate **SPoT** on the test sets by comparing for each text plan:

- HUMAN: The score of the top-ranked sentence plan.
- BOTH: The score of the **SPR**'s selected sentence when trained on both feature sets.
- SP-TREE: The score of the **SPR**'s selected sentence when trained only on sp-features.
- DSYNTS: The score of the **SPR**'s selected sentence when trained only on DSyntS-features.
- RANDOM: The score of a sentence plan randomly selected from the alternate sentence plans.

Figure 6 shows the distributions of rankings for the highest ranked sp-tree for each of the 98 text plans, for HUMAN, BOTH, SP-TREE, DSYNTS, and RANDOM. The x-axis is labeled with scores, and the y-axis with number of turn instances. Each data point represents the number of turn instances for which the chosen utterance of the particular distribution received a score equal to or higher than the score on the x-axis. Thus, the better the distribution, the flatter the curve; curves that fall quickly as the score increases represent bad strategies. The best scores (HUMAN) provide a topline for the **SPR** (no distribution can be better with our experimental design), while RANDOM provides a baseline. The HUMAN distribution shows that nearly all of the turns had at least one sentence plan which scored 4 or better. The RANDOM distribution approximates the rankings for all sentence plans for all examples.

Because each turn is used in some fold of the 5-fold cross validation as a test element, we assess the significance of the ranking differences with paired t-tests. A paired t-test of SP-TREE and BOTH shows that there are significant differences in performance ($p = 0.03$). The mean of SP-TREE is 4.40 as compared with the mean of BOTH of 4.55, for a mean difference of 0.15 on a scale of 1 to 5. Both SP-TREE and BOTH have a median of 5 (i.e., both make an optimal choice over half the time). A paired t-test of DSYNTS to SP-TREE shows that there are also significant differences in performance ($p = 0.01$). The mean of DSYNTS is 4.11 as compared with the mean of SP-TREE of 4.40, for a mean difference of 0.29. The median of the DSYNTS distribution is 4.5. Finally, we compare DSYNTS to the RANDOM baseline. Again, we see significant differences in performance when we apply a paired t-test ($p < 0.01$).

We then examined the rules that the **SPR** learned in training (and the resulting RankBoost scores), for both the general

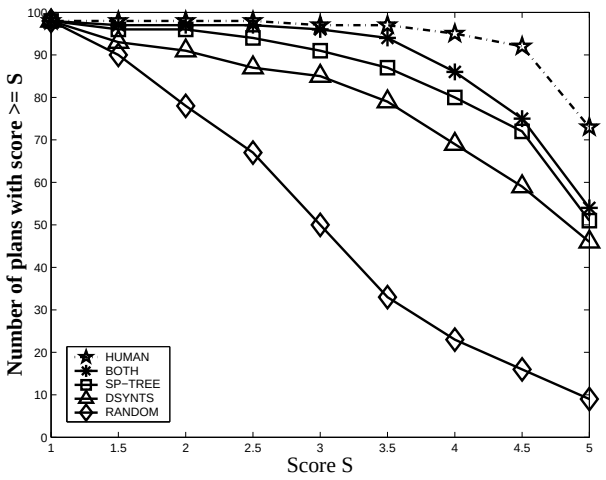

Figure 7: Distribution of rankings for HUMAN, BOTH, SP-TREE, DSYNTS, and RANDOM

N	Condition	α_s
1	LEAF_IMPLICIT_CONFIRM ≥ 0.5	1.17
2	RULE_ANC_EXPLICIT_CONFIRM*MERGE_GENERAL ≥ 0.5	1.1
3	RULE_ANC_REQUEST_DEPART_TIME*SOFT_MERGE_GENERAL ≥ 0.5	0.52
4	RULE_ANC_IMPLICIT_CONFIRM*MERGE_GENERAL ≥ 0.5	0.39
5	RULE_ANC_REQUEST_DATE*SOFT_MERGE_GENERAL ≥ 0.5	0.38
6	RULE_TVL_RANDOM_CUEWORD*IMPLICIT_CONFIRM ≥ 0.5	-0.17
7	PERIOD_MAX ≥ 2.5	-0.09
8	RULE_TVL_IMPLICIT_CONFIRM ≥ 1.5	-0.08

Figure 8: The eight rules generated using the sp-features on the first test fold which have the largest positive impact on the final RankBoost score (above the double line) and the largest negative impact on the final RankBoost score (below the double line). α_s represents the increment or decrement associated with satisfying the condition.

and lexicalized features. Figures 8 and 9 show the rules that had the largest impact on the RankBoost score which determines the ranking of alternative sp-trees (for the first test fold). We discuss some particular rule examples here to illustrate how **SPoT** works. Rule (1) in Figure 8 suggests that the implicit confirm should be done first, and Rule (2) states that a tree where MERGE-GENERAL is the immediate ancestor of an explicit confirm gets a large increase in ranking. Rule (6) says that an implicit confirm with a cueword is penalized.

The DsyntS rules in Figure 9 have stronger negative values, indicating lexical constructions that humans strongly dislike. For example, the SENT_ANCESTOR_AND*LIKE rule (rule (1)) is often found in DsyntSs with realizations such as *Where would you like, and going to San Jose, to go?*. Rule (2) states that two or more pronouns (in our domain, *you* and empty pronouns) in the same utterance results in a penalty. Rule (7) penalizes arguably irritating overuse of cue words (such as *Now, you are leaving at 4. Now, where are you going to?*).

5. Discussion

Our results show that when both domain dependent and domain independent features are utilized in training, that **SPoT** selects sentence plans that on average are only 5% worse than those selected as the best by human judges and 36% better than those selected by a randomized **SPR**. The performance of **SPoT** trained on the more general sp-features was significantly better than that trained on DSyntS-features, and a combination of the two had the best performance. Thus while **SPoT** has good performance without utilizing the lexicalized/domain-specific features, it takes advantage of them when they are provided. Both types of features lead to general rules that don't utilize parameterized features that are dependent on particular instantiations of travel plans. In that respect, the DSyntS-features serve as a "style" guide, while the sp-features measure how well the utterance is structured. Also, the negative alpha values indicate that the DSyntS-features tend to lower the rank of the lower quality utterances, while the sp-features increase the rank of the higher quality variants.

N	Condition	α_s
1	SENT_ANC_AND*LIKE ≥ 0.5	-0.89
2	SENT_TVL_PRONOUN ≥ 1.5	-0.81
3	SENT_ANC_COMMA*AND*LIKE ≥ 0.5	-0.69
4	SENT_TVL_IN1 ≥ 0.5	-0.54
5	SENT_ANC_SAN_JOSE*TO1*GO ≥ 0.5	-0.51
6	SENT_SIS_WANT ≥ 0.5	-0.37
7	SENT_TVL_NOWCOMMA ≥ 1.5	-0.32
8	SENT_TVL_GO*PRONOUN*WHERE*TO1 ≥ 0.5	0.31
9	SENT_TVL_PERIOD*WANT*NEED ≥ 0.5	0.29

Figure 9: The nine rules generated using the DSyntS-features on the first test fold which have the largest negative impact on the final RankBoost score (above the double line) and the largest positive impact on the final RankBoost score (below the double line). α_s represents the increment or decrement associated with satisfying the condition.

6. References

[1] M. Collins. Discriminative reranking for natural language parsing. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2000.

[2] Y. Freund, R. Iyer, R. E. Schapire, and Y. Singer. An efficient boosting algorithm for combining preferences. In *Machine Learning: Proceedings of the Fifteenth International Conference*, 1998. Extended version available from http://www.research.att.com/ schapire.

[3] B. Lavoie and O. Rambow. A fast and portable realizer for text generation systems. In *Proceedings of the Third Conference on Applied Natural Language Processing, ANLP97*, pages 265–268, 1997.

[4] I. A. Melčuk. *Dependency Syntax: Theory and Practice*. SUNY, Albany, New York, 1988.

[5] O. Rambow and T. Korelsky. Applied text generation. In *Proceedings of the Third Conference on Applied Natural Language Processing, ANLP92*, pages 40–47, 1992.

[6] J. Shaw. Clause aggregation using linguistic knowledge. In *Proceedings of the 8th International Workshop on Natural Language Generation*, 1998.

[7] M. Walker, O. Rambow, and M. Rogati. Spot: A trainable sentence planner. In *Proceedings of the North American Meeting of the Association for Computational Linguistics*, 2001.