Generating Canonical Example Sentences using Candidate Words [*]

John Dowding Gregory Aist Beth Ann Hockey Elizabeth Owen Bratt
RIACS
NASA Ames Research Center
Moffett Field, CA 94035
{jdowding,aist,bahockey}@riacs.edu
Center for the Study of Language and Information
Stanford University
Stanford, CA
ebratt@csli.stanford.edu

Abstract

Situations where a spoken dialogue system cannot interpret a user's utterance are a major source of frustration in human-computer spoken dialogue. Current spoken dialogue systems generally respond with an unhelpful "I'm sorry, I didn't understand," or something similarly uninformative. Recent work on Targeted Help has shown that giving users more appropriate feedback makes systems easier to learn and improves performance. In particular, Targeted Help can give the user an appropriate within-domain example sentence to help the user more quickly learn the system's lexical and grammatical coverage. This paper addresses the generation problem of how to find such an example sentence. We present and evaluate four algorithms for solving this generation problem: an Iterative-Deepening (ID) algorithm, an A^* algorithm, a combined A^*-ID algorithm, and an anytime algorithm.

Introduction

The goal of Targeted Help in a spoken dialogue system is to provide the user with more specific information in cases in which the system is not able to understand their speech. For instance, if the user is unfamiliar with a push-to-talk device, the system may recognize that the user was already speaking when the push-to-talk button was pressed, and the system might respond with *Please be sure that the button is pressed before you begin speaking*. Previous work comparing grammar-based language models with statistical language models (Knight *et al.* 2001) has shown that grammar-based language models show superior recognition accuracy for within-grammar utterances compared to a statistical language model. Systems that use grammar-based language models will then benefit from teaching the user the bounds of the system quickly, and encouraging them to use within-grammar utterances. Recent work (Hockey *et al.* 2003) has shown that Targeted Help leads to increased task-success and decreased time-to-complete on a urban patrol and search and rescue task in a mixed-initiative dialogue system to control a simulated robotic helicopter.

There are a number of strategies for what type of Targeted Help system response might be produced. The previous example illustrates a diagnostic response. Another possibility is to produce an in-coverage example sentence similar to what the user said. Some obvious questions are: 1) what can be known about the user's utterance if the system is not able to understand it? and 2) in what way can the generated utterance be made similar to the user's utterance? Even when the user produces an out-of-coverage utterance they are likely to produce some in-coverage words. Our Targeted Help system runs a fall-back speech recognizer that is being driven by a category-based statistical language model. When the grammar-based recognizer fails, the system looks for within-domain words in the recognition hypothesis from the fall-back recognizer. This gives us a set of target words (potentially with word-confidence scores), and we then try to generate a grammatical example utterance containing those words. For example, if the user says something like *I'd like the pressure at the commander*, and the fall-back recognizer detected the words *pressure* and *commander*, the generator could provide an example grammatical utterance like *measure the pressure at the commander's seat*. This paper focusses on algorithms for generating this type of in-coverage example.

The choice of an ideal example sentence could conceiveably take into account information from a wide variety of sources, including discourse history, user model, and pedagogical strategy. This could lead to constraints on not only what words should be included in the example, but also what syntactic structures, semantic representations, and word order should be used. For this paper, we have settled on an initial simplification of this task, using only a set of desirable candidate words. Our definition of this generation task is then:

> Given a grammar G, and a set of target words T, find a word string $W = W_1 ... W_n, \quad W \in L(G)$ such that $T \subset \{W_1, ..., W_n\}$.

This paper presents and evaluates four algorithms for solving this generation problem: an Iterative-Deepening (ID) algorithm, an A^* algorithm, a combined A^*-ID algorithm, and an anytime variant of the A^*-ID algorithm. We will discuss some potential extensions to these algorithms to incorporate more information into the generation task under Future Work.

[*]The research reported in this paper performed at RIACS was supported under NASA Cooperative Agreement Number NCC 2-1006.

Evaluation Domains

The algorithms described in this paper will be evaluated using unification-based grammars developed in the Gemini formalism (Dowding *et al.* 1993) for two spoken dialogue systems: the PSA simulation system (Rayner, Hockey, & James 2000) and the WITAS system (Lemon *et al.* 2001). Both systems are spoken dialogue interfaces to control semi-autonomous robots. The PSA system controls a simulation of the Personal Satellite Assistant (PSA), a mobile robot under development at NASA's Ames Research Center, and intended for deployment aboard the Space Shuttle and International Space Station. The WITAS system controls a simulation of a semi-autonomous helicopter engaged in urban patrol and search and rescue tasks. Both systems have the property that they use a single unification grammar for the three purposes of parsing/interpreting the user's utterances, generating the system's responses, and acting as a grammar-based language model in the speech recognizer. Generating the system's responses is carried out using a variant of Semantic Head-Driven Generation (Shieber *et al.* 1990). The algorithms presented in this paper let us use the unification grammar for a 4th purpose, to generate canonical example sentences.

Restricting Generation to Canonical Sentences

Although the language that a system can hear in a spoken dialogue system and the language that it can (or should) generate are naturally related, they need not be identical. For instance, for the sake of robustness, a system may be able to parse telegraphic or fragmentary user utterances, but it may choose to always respond to the user in fully grammatical utterances. Gemini provides a feature that allows us to effectively subset the grammar into the parts that we want to make available for generation, and the parts we do not. Our two evaluation grammars have been subsetted in this way into *canonical* and *non-canonical* subsets. The grammar writer needs only mark **canonical=no** on those grammar rules and lexical items that they want to exclude from generation, and the generator specifies **canonical=yes** on the call to the top-level category.

The remainder of this paper will not make use of any Gemini-specific capabilities. The internal representations of these Gemini grammars have been translated into the Definite Clause Grammar (DCG) formalism (Pereira & Warren 1980), and the algorithms described here operate on these DCG representations, so the algorithms should be applicable to other unification-based grammar formalisms. Prolog source code for all of these algorithms will be made available on an open-source license basis at http://www.OpenNLP.com.

Iterative Deepening Algorithm

The first algorithm we tried used a simple iterative deepening (Korf 1985) (ID) approach in a generate-and-test program. Iterative deepening simulates a breadth-first search in a depth-first way by putting a bound on the amount of computation done in each depth-first step, and iteratively increasing that bound. Effectively, ID as a generator finds all the grammatical parse trees with 1 node, followed by all with 2 nodes, then 3 nodes, etc. As ID is generating these parse trees, each is tested to see if it contains the target words. We built an iterative deepening generator first in part because it was very easy to build (taking just a few minutes), and we were hopeful that the inherent speed of Prolog's depth-first engine would compensate for the unguided nature of the search. Implementing ID efficiently for DCGs is easy because the additional arguments needed to enforce ID's computational bounds can be inserted in the DCG rules at compile time, in a way exactly analogous to the way that DCG string positions are automatically added to a DCG rule before it is converted to Prolog's clausal form. Thus, the ID version of the grammar is running in fully-compiled Prolog clausal form with no interpreter overhead.

"The A^* algorithm, described by Hart, Nilsson, and Raphael (1968), addresses the problem of finding a minimal-cost path joining the start node and a goal node in a state-space graph. ... The algorithm used by A^* is an *ordered state-space search*. ... Its distinctive feature is its definition of the *evaluation function, f**. As in the usual ordered search, the node chosen for expansion is always one at which f* is minimum.
Since f* evaluates nodes in light of the need to find a minimal-cost solution, it considers the value of each node *n* as having two components: the cost of reaching *n* from the start node and the cost of reaching a goal from node *n*. Accordingly, *f** is defined by *f*(n) = g*(n) + h*(n)*, where *g** estimates the minimum cost of a path from the start node to node *n*, and *(h*)* estimates the minimum cost from node *n* to a goal. The value *f*(n)* thus estimates the minimal cost of a solution path passing through node *n*. The actual costs, which *f**, *g**, and *h** only estimate, are denoted by *f*, *g*, and *h*, respectively. It is assumed that all arc costs are positive.
The function *g**, applied to a node *n* being considered for expansion, is calculated as the actual cost from the start node *s* to *n* along the cheapest path found so far by the algorithm.
... The function *h** is the carrier *heuristic information* and can be defined in any way appropriate to the problem domain. For the interesting properties of the A^* algorithm to hold, however, *h** should be nonnegative, and it should never overestimate the cost of reaching a goal node from the node being evaluated."

Figure 1: A^* Definition - Barr and Feigenbaum (1981) pp.64-65

A^* Algorithm

Our next attempt was motivated by the observation that the simple generate-and-test method was not taking advantage of the fact that we know we are looking for a particular set of target words. We designed an A^* algorithm (see Figure 1) using a cost function that would prefer hypotheses containing target words to those that don't. Achieving good performance of an A^* algorithm depends on the quality of the scoring function f^*, in particular on the choice of h^*. To operationalize the constraint "including all of the words in the target set", we defined a notion of *fruitful*. A nonterminal is *fruitful* if it dominates one or more of the target words. Each nonterminal in a derivation is scored both for the number of target words it may dominate (its number of fruit), and minimum number of rule applications it will take to consume the nearest fruit. We pre-compute a table $< C, W, D >$ where a nonterminal C can derive a word W in D rule applications. Since this is a unification-based grammar, the nonterminal C is a Prolog term, and the table is maintained with only the most-general instantiations of each nonterminal, with the exception that a nonterminal C that is subsumed by a nonterminal C' is retained in the table if the depth of C is less than the depth of C'. When matching a particular nonterminal against this table while estimating $h(hyp)$, we find the minimum depth in the table that unifies with that nonterminal.

We explored several definitions of f^*, for both admissible and inadmissible A^* searches. The results given in Table 1 use this definition:

$$\begin{aligned} f^*(n) &= g^*(n) + h^*(n) \\ g^*(n) &= \text{the number of steps in the derivation so far} \\ h^*(n) &= max_{nt \in derivation, w \in T} < nt, w, d > \\ &\quad +(N-1) \end{aligned}$$

This definition for f^* uses an estimate h^* that never overestimates the number of steps to reach a final derivation, so this yields an admissible search. The motivation behind this estimate is to first find the target word that requires the most steps to derive, since the estimate will require at least that many steps. To this we add 1 for every other nonterminal in the derivation, since they will require at least 1 step each to derive a final terminal string.

The implementation of this A^* algorithm proceeds as follows. We maintain a priority queue of states, sorted on their value for f^*. Each state is a 4-tuple $< S, D, G, TW >$, consisting of a score S, a current derivation D, the number of derivational steps G and the set of target words TW. The algorithm proceeds:

- Initialize a priority queue with a start state defined as a 4-tuple $< Score, [S], 0, TW >$ with an initial score of $h([S])$, a derivation starting at the start symbol S, the initial value for G = 0, and the set of target words TW.
- Iterate on:
  - remove the highest state from the priority queue
  - choose a nonterminal to expand, preferring a fruitful nonterminal with minimal depth
  - find new derivations by expanding the chosen nonterminal by all matching grammar rules
  - compute the set of next states based on these derivations, updating Score, Derivation, G and TW
  - add the new states to the priority queue
- Terminate when the top state in the priority queue has $TW = \emptyset$ and its derivation contains no nonterminals.

It is not clear that the admissibility property is critical for this approach. We have experimented with several inadmissible heuristic functions, and find that they appear to provide acceptable solutions, with some improvement in performance.

Combined A^*-ID Algorithm

The termination condition for the A^* algorithm described above is when the word string has been completely generated, continuing the A^* search even after all of the target words have been consumed. The A^* algorithm was motivated as a way to carry out a more directed search. Unfortunately, once the target words have all been found, they provide no more direction to the search. We thus devised a variant of this algorithm where the A^* algorithm terminates once all the target words have been found. The end product of this A^* search is a derivation that is a combination of terminal and nonterminal symbols. We produce the final word string by using the ID generator to find word strings for the remaining nonterminals. To retain admissibility for the A^*-ID, it is critical that the remaining nonterminals be generated as a conjunction of nonterminals in a single ID search, since the nonterminals may share variables that effect the joint solution.

Anytime Algorithm

So far we have written a generator that would generate utterances quickly given a single target word, and successively more slowly with additional target words. We can convert this into an anytime algorithm, defined as "An algorithm that returns the best answer possible even if it is not allowed to run to completion, and may improve on the answer if it is allowed to run longer." (Howe 1998).

The idea is to allow the algorithm to take the maximal amount of time (in this case, 30 seconds), and to produce the best answer possible by first finding a solution using 1 target word, then using 2 target words, etc. until the allowed time has elapsed. The best solution found when time elapses is returned as the final answer. The anytime variant works best when the target words are priority ordered.

In our anticipated deployment in Targeted Help, however, not all words will be treated equally. In particular, our most important goal is to include the main verb in the generated sentence; the object is of second importance; other words after that. Therefore, we wrapped our A^* generator into an anytime algorithm as follows. Given a list of words [MainVerb, Object—Rest]: First, generate with the empty list ([]). (This just returns a default example, and is instantaneous.) Second, generate with [MainVerb]. Third, generate with [Object]. Continue to generate, adding one word from Rest each time.

Evaluation

Our first experiment was to compare the performance of ID, A^* and A^*-ID. Ideally, we would have liked to run a

	PSA		
	ID	A^*	A^*-ID
0 Words	0ms.	150ms.	10ms.
1 Word	84%	96%	100%
	(211/0/39)	(241/0/9)	(250/0/0)
	2401ms.	969s.	458ms.
2 Word	27%	41%	41%
	(27/0/73)	(41/15/44)	(41/15/44)
	8312ms.	6205ms.	5932ms
3 Word	1%	6%	6%
	(1/0/99	(6/35/59)	(6/35/59)
	11066	6900ms.	6890ms.
4 Word	0%	1%	1%
	(0/0/100)	(1/37/62)	(1/37/62)
	—	28391ms.	29291ms.
	WITAS		
	ID	A^*	A^*-ID
0 Words	0ms.	70ms.	0ms.
1 Word	94%	94%	94%
	(367/0/24)	(367/24/0)	(367/24/0)
	206ms.	351ms.	304ms.
2 Word	21%	41%	44%
	(21/0/79)	(41/42/17)	(44/42/14)
	5688ms.	6732ms.	6159ms
3 Word	1%	16%	20%
	(1/0/99	(16/41/43)	(20/41/39)
	27670ms	13868ms.	14727ms.
4 Word	0%	4%	5%
	(0/0/100)	(4/64/32)	(5/64/31)
	—	28391ms.	29291ms.

Table 1: Timing Results for ID, A^*, and A^*-ID - Numbers in Parentheses indicate (Solved/Failed/Timed Out)

speech recognizer with a statistical language model on data collected from real users of a targeted help system, but that data is not yet available. To simulate that, we generated a set of target word sets of increasing sizes. For the sake of completeness, we include the amount of time to generate a grammatical sentence from a 0-word target set. For the case of 1-word target word sets, we considered singleton sets for each lexical item in the vocabulary. For 2-word sets and higher, we randomly generated 100 2 word (and higher) target words sets by randomly selecting target words from the vocabulary for each grammar. The results of this experiment are given in Table 1. All timing numbers given in the table were run on a 1.6GHz P4 laptop with 1GB of memory and running Windows 2000 and SICStus Prolog 3.10.0. We ran the experiments with a time-out of 30 seconds. We report:

- the percentage of the target-word sets that each algorithm was able to find example sentences for.
- the break-down (Success/Failure/TimedOut) for each test. Failure indicates that there is no canonical sentence in $L(G)$ containing those target words.
- the average time for the non-timed-out instances of the test set. We excluded the times for the timed-out instances since they take a uniform 30 seconds each, and we didn't want the choice of timeout interval to be reflected in the timing numbers.

As can be seen, all 3 algorithms did relatively well at generating grammatical sentences from a 1-word target set, with the A^*-ID algorithm performing the best, and the ID algorithm performing the worst. However, as the target set size increases, the performance of all algorithms drops markedly. It may be surprising that so many of the larger target-word set instances failed to generate an example, or that, for WITAS, even 24 of the singleton target-word sets failed to generate an example. Keep in mind, however, that the grammar-writer has excluded certain lexical items and grammar rules from generation as being non-canonical, and that the target-word sets were selected from the entire vocabulary, not just the canonical vocabulary. Note also that the apparent advantage of $A*$-ID over A^* is greatest for the PSA grammar where there are no failures. In failure cases, A^* and A^*-ID will usually behave identically, and fail before getting to an ID search.

We ran a second experiment to evaluate the anytime algorithm. Since the anytime algorithm always runs until the entire time period has elapsed, using time as a measure of performance is not appropriate. Given the results above, we were interested in the performance of the algorithm in cases where we know that a candidate solution exists. We limited the dataset to target sets containing words that we knew could be found together in a grammatical sentence. We picked 100 grammatical sentences from the PSA grammar, and ran the anytime algorithm on the words from each sentence in random order. The results are show in a histogram in Figure 2, showing how many target words were taken into consideration for each sentence at the point where the time bound was met. In this figure, we can see that the anytime algorithm is usually able to use target-word-set sizes in the range 3-5. If the execution time of the $A^* - ID$ algorithm was critically bound by the size of the target word set, then we would expect to see most sentences only being generated from a small number of target words. The figure shows that the opposite is true: when a sentence is within grammar, the A^*-ID algorithm is able to find an example sentence using relatively large target word sets.

Discussion and Future Work

There are a number of plausible extensions to the anytime algorithm. There is no requirement that the target word sets be unordered. We could use recognizer confidence scores as one way to prefer example sentences containing some words over others. We could also use part-of-speech information to prefer content words (nouns and verbs) over function words.

We can also extend the notion of *fruit* to include not just specific words, but specific categories. That is, to look for example sentences with additional constraints on the result to be of a certain grammatical form. For instance, if the dialogue manager knows that it has asked a question that would typically be answered with an isolated noun-phrase fragment, it could add the request that the example sentence be in the form of a noun phrase. Similarly, in order to speed up system-user discussions about future events, we might

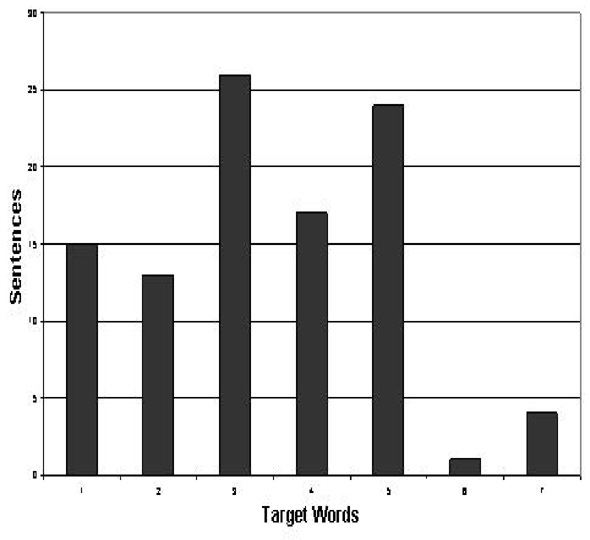

Figure 2: Anytime - PSA

want the system to demonstrate to the user that the system can handle relative clauses.

There is a striking similarity between the problem of generating good example sentences for Targeted Help, and certain robust interpretation techniques. Minimal Edit Distance parsing (Aho & Pederson 1972) appears on its surface to be very similar. The problem they solve is, given a word sequence W, find another word string $\hat{W}$, such that $\hat{W} \in L(G)$ and W can be derived from $\hat{W}$ with a minimal number of insertions, deletions, and substitutions. The principal difference is that we are considering arbitrary word order on the target word sets, while they do not. This similarity should perhaps not be surprising, since, if we had very high confidence in the quality of our example sentences, it is a very small step to go from responding *You can say sentences like "measure the pressure at the commander's seat"* to *Did you want me to measure the pressure at the commander's seat?*. The fact that there is an elegant $O(n^3)$ solution to the Minimal Edit Distance parsing problem suggests that perhaps there is a polynomial solution to this generation problem as well.

Conclusion

This paper presents a novel and important problem, generating good example sentences for use in Targeted Help for spoken dialogue systems. We present 3 admissible algorithms addressing this problem, and compare their relative performance. A 4th algorithm is provided that shows that an example can almost always be generated from the target words when such an example exists in the grammar.

References

Aho, A., and Pederson, T. 1972. A minimum distance error-correcting parser for context-free language. *SIAM* 1.

Barr, A., and Feigenbaum, E. 1981. *The Handbook of Artificial Intelligence*. William Kaufman, Inc.

Dowding, J.; Gawron, M.; Appelt, D.; Cherny, L.; Moore, R.; and Moran, D. 1993. Gemini: A natural language system for spoken language understanding. In *Proceedings of the Thirty-First Annual Meeting of the Association for Computational Linguistics*.

Hockey, B.; Lemon, O.; Campana, E.; Hiatt, L.; Aist, G.; Hieronymus, J.; Gruenstein, A.; and Dowding, J. 2003. Targeted help for spoken dialogue systems: intelligent feedback improves naive users' perfomance. In *Proceedings of the European Chapter of the Association for Computational Linguistics 2003 (accepted for presentation)*.

Howe, D. 1998. Free online dictionary of computing. *http://wombat.doc.ic.ac.uk/foldoc*.

Knight, S.; Gorrell, G.; Rayner, M.; Koeling, R.; and Lewin, I. 2001. Comparing grammar-based and robust approaches to speech understanding: A case study. In *Proceedings of Eurospeech 2001*.

Korf, R. 1985. Depth-first iterative deepening: An optimal admissible tree search. *Artificial Intelligence* 27:97–109.

Lemon, O.; Bracy, A.; Gruenstein, A.; and Peters, S. 2001. A multi-modal dialogue system for human-robot conversation. In *Proceedings of North American Association for Computational Linguistics (NAACL 2001)*.

Pereira, F., and Warren, D. 1980. Definite clause grammars for language analysis – a survey of the formalism and a comparison with augmented transition networks. *Artificial Intelligence* 13:231–278.

Rayner, M.; Hockey, B.; and James, F. 2000. A compact architecture for dialogue management based on scripts and meta-outputs. In *Proceedings of ANLP 2000*.

Shieber, S.; van Noord, G.; Moore, R.; and Pereira, F. 1990. A semantic head-driven generation algorithm for unification grammars. *Computational Linguistics* 16(1).