Let's Go Public! Taking a Spoken Dialog System to the Real World

Antoine Raux, Brian Langner, Dan Bohus, Alan W Black, Maxine Eskenazi

Language Technologies Institute, Carnegie Mellon University, Pittsburgh PA, USA
{antoine,blangner,dbohus,awb,max+}@cs.cmu.edu

Abstract

In this paper, we describe how a research spoken dialog system was made available to the general public. The Let's Go Public spoken dialog system provides bus schedule information to the Pittsburgh population during off-peak times. This paper describes the changes necessary to make the system usable for the general public and presents analysis of the calls and strategies we have used to ensure high performance.

1. Introduction

The Let's Go bus information dialog system [1] was created as a test bed for spoken interaction with extreme user populations. Specifically, the focus was on making spoken dialog systems more accessible to the elderly and non-natives. Earlier efforts in this work were on speech synthesis modifications to aid the elderly understand synthetic speech [2] and changes in speech recognition to aid non-natives [3].

The Port Authority of Allegheny County (PAAC) provided data for the system both in the form of bus schedules and recorded dialogs between callers and human operators. In the fall of 2004, the management of PAAC called the Let's Go experimental system and found that it could correspond to their users' needs for information outside of the hours when human operators answer the phone lines. The decision was made to make Let's Go publicly available in early March 2005 for two main reasons: this would allow the Let's Go group to collect speech data from real users to train the system; and the group could carry out a series of tests of the system with real users.

Experimental dialog systems are very different from commercial systems. The changes that are made in order to accommodate public use would initially seem to render the system less useful for experimental studies. In this article, we explain the changes we have made to the Let's Go system to make it useful for the general public while retaining aspects that will allow us to continue to experiment with dialog systems in general and for extreme user populations. We describe the performance of this baseline system, and analyze the understanding errors, and the strategies used to overcome them.

This analysis is based on a corpus of dialogs we have collected since the system went "live", amounting to an average of 40 calls per day.

2. System Architecture, Components, and Adaptations For Public Use

2.1. Speech Recognition

We use the CMU Sphinx2 speech recognizer with gender-specific telephone quality acoustic models from the Communicator system [4]. We run both male and female recognizers in parallel and automatically select the best result. This method, as others have found, improves our recognition accuracy.

Our original language model was a state-specific model trained on past calls to the mixed-initiative system. Due to the differences between the two systems, we were expecting a mismatch between the old language models and user utterances in the new system. Moreover, the modifications we made to the dialog task generated new states in the dialog for which no data was yet available. We initially resorted to hand-coded Finite State Grammars for confirmations and neighborhood names but we found that recognition accuracy (and particularly false rejection) was not satisfying so we finally settle on tri-gram language models trained on artificial corpora generated using simple templates and information from the database (for neighborhood names).

In addition to speech recognition, we allowed the user to use touch tones for Yes/No answers for explicit confirmation and to ask for help (which they could also do at any time by simply saying "Help").

2.2. Dialog Manager

Dialog management in the Let's Go system is based on the RavenClaw architecture [5], for dialog management. RavenClaw features a task-independent dialog engine that carries out a dialog according to a given task specification.

Our initial system was designed for mixed-initiative, fairly open-ended dialogs. Although good for exploring and experimenting with natural spoken language interactions, this approach makes the system more fragile in the presence of less-than-optimal conditions. Although the RavenClaw framework already supports a variety of recovery strategies, we modified our baseline system towards a very conservative and cautious approach to dialog. This did not require any changes to RavenClaw architecture itself, but a new task description which encodes a fairly linear, system-initiative dialog that asks the user for three or four concepts sequentially: an optional bus route number, a departure place, a destination and a desired travel time. Each concept is explicitly confirmed so it is rare that the dialog advances without the system getting the correct information from the user. The drawback of this conservative approach is that users can get "stuck" when the system fails to recognize one concept. Moreover, the pace of the dialog is somewhat slow and, even without recognition errors, expert users might find it frustrating to have to provide and confirm one concept at a time. Nevertheless, we opted for this solution so as to maximize the chances of task success.

In addition to the "normal" flow of the dialog, we explicitly coded subdialogs dealing with bus routes or areas of the city for which the system does not provide schedule information. In these cases the system simply informs the user of the limited coverage of the current version and allows them to start a new query. We also added a back-off strategy triggered by 2 successive failures to get the departure place. This strategy first asks for the neighborhood from which the user is leaving, and then ask for the specific stop in this neighborhood (or informs them that the neighborhood is not covered by the system). More details about this strategy and the ensuing user behavior are given below.

2.3. Backend Manager

The backend of the system is primarily a database of bus schedules and routing information provided by PAAC. The PAAC system consists of 14,983 stops (although some stops have multiple names). The stop names were regularized, expanding abbreviations, allowing us to say that "5th Ave", and "Fifth Avenue" all reference the same street, but maintaining the distinction with "5th Street". There are 2428 routes (including variations of routes

according to time of day or week).

2.4. Language Generation

For language generation, we are using Rosetta, which is a language generation toolkit originally designed for the CMU Communicator. Rosetta is capable of generating utterances from templates, filling in slots with information received from the dialog manager. It can also randomly select from a list of templates for a given response. The generated utterances are then sent to the text-to-speech engine for synthesis.

In addition to changing system utterances to account for the difference between a user-initiated and a system-initiated dialog, we also modified the language generation to produce shorter utterances in general, and longer, more detailed help utterances.

2.5. Speech Synthesis

As most of the spoken output in such a system is constrained, and high quality output is required, we followed the techniques in Limited Domain Synthesis [6], to build a unit selection concatenative speech synthesis voice specifically designed for this domain.

Although much of the output follows well defined templates, or informational prompts, with almost 15,000 bus stops it is not possible to record even one example of each stop name. Therefore in addition to the standard templates, bus numbers, informational prompts, we also designed a sub-corpus that maximized diphone coverage over the 15,000 bus stop names. The voice is good almost all the time.

For the Public system we had to spend time tuning it to respond fast enough to keep the dialog going, and although the voice can say anything, even things outside the domain, we further handcrafted some of the prompts to avoid poorly synthesized examples.

3. Corpus Description

3.1. Data Collection

The Let's Go Public system runs every night from 7pm to 7am on weekdays and from 6pm to 8am on weekends and holidays. During these hours, when PAAC users call customer service, they first get to choose between schedule information and other types of requests (e.g. complaints) through a standard touch-tone interface. After selecting schedule information, they are informed that operators are not working at the time and are given the possibility to transfer to the Let's Go system by pressing "one". This message also lists the 10 routes that are currently covered by the system. This initial interaction filters out some of the requests that the system can not handle.

In the first three weeks of operation, we gathered a total of 614 dialogs (excluding calls which do not contain any speech directed at the system), containing 7936 user turns. All these dialogs were manually transcribed and then checked by a second annotator. Transcriptions include noise labels as well as a variety of tags for asides and other speech not directed to the system. Table 1 shows some basic corpus statistics

	Whole Corpus	Turns with Speech
# dialogs	627	614
# turns	9162	7936
Turns/dialog	12.9	14.6
Word Error Rate	68%	60%
Understanding Error Rate	55.0%	48.1%
Task success	43.3%	43.6%

Table 1: *Overview of the Let's Go Public corpus.*

3.2. System Performance

3.2.1. Word Error Rate

While we can assume that most users were calling to fulfill a genuine information need, we found a large range of attitudes toward the system, from amusement, to infinite patience, and sometimes sarcasm. The conditions in which people called were also very diverse, some calls being made from a quiet indoor place, while others were done from cell phones while on the street, and others from noisy rooms (due to background conversations, loud television, babies crying, etc).

The raw word error rate (WER) computed from the transcriptions is 68%, which is significantly higher than what we have previously observed using the same engine and acoustic models on more controlled conditions (17% for native and 43% for non-native speakers in [7]). Part of the problem is background noises and utterances by the main speaker directed at another person rather than at the system. Yet, even when taking these utterances out of the computation, the WER is still 60%.

3.2.2. Task Success

Because each caller's actual information need is unknown, many calls cannot be easily labeled as success or failure. Two of the authors inspected the dialogs (with access to the recordings, the transcriptions and all of the system information) and marked them as success if they felt that the system did answer the users need as they understood it. This includes some tolerance if for example the system misrecognized the departure stop for a nearby stop, which the user accepted. All calls in which the system did not give any information to the user were labeled as failures, although the reason for users hanging up in the middle of a call can be unrelated to the quality of the dialog (e.g. their bus arrives). Finally, we noticed several cases where, when having a difficult conversation with the system, users hung up and called back right away. The kappa coefficient for the agreement between the two annotators was 0.75. Since one of the annotators labeled a larger portion of the dataset (463 dialogs), we use their labels as our "truth" for task success. The overall success rate is 43.3%, 43.6% when excluding sessions that did not contain any system-directed speech.

3.2.3. Dialog Length

The mean number of turns per call is 12.9, with a relatively large standard deviation of 11.5 turns. 22% of the calls are shorter than 6 turns (the minimum necessary to get schedule information, including confirmations), and 16% longer than 20 turns. In addition, although the system does understand "start over" as a request to restart the dialog from scratch, certain users preferred to hang up and call back immediately when facing many misunderstandings. This resulted in "splitting" certain queries over several calls.

3.3. Understanding Errors

3.3.1. Definitions

To be successful, calls require three or four pieces of information from the user: a departure stop, a destination, a travel time, and, optionally a bus route. Moreover, the names with which users refer to places might not match the names used in the schedule database (although we did extend the original database with landmarks). Finally, while the introductory prompt explicitly listed the routes covered by the system, we still encountered a significant number of users asking for uncovered routes and areas of the state.

In order to quantify understanding at the turn level, we define the following terms:

- non-understanding: a turn where the system could not extract any meaning from the user utterance

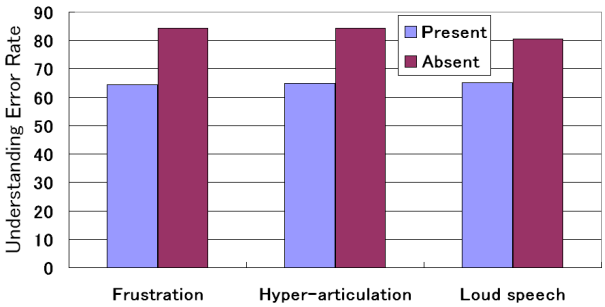

Figure 1: *Speaking styles and understanding error rate following non-understandings.*

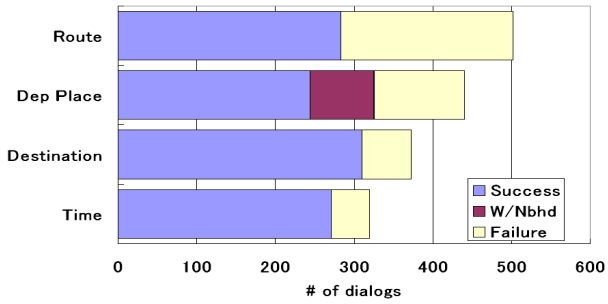

Figure 2: *Success rate of subdialogs for the acquisition of the four key concepts.*

- misunderstanding: a turn where the system wrongly understood at least one concept from the user utterance, either by inserting a concept or getting the wrong value for a concept (simple concept deletions, which are usually less harmful to the dialog, are not considered misunderstandings).
- understanding error: a turn that is either a non-understanding or a misunderstanding
- understanding error rate: the ratio of understanding errors over the total number of turns

The overall understanding error rate is 55% on the whole corpus, 48.1% when ignoring non-speech utterances.

3.3.2. Speaking Styles

One author manually labeled a subset of 991 turns for hyper-articulation and loud speech (as compared to the other utterances from the same call). We found 10.2% of turns hyper-articulated and 11.2% turns of loud utterances, which span respectively 57.8% and 46.4% of the dialogs. Another author labeled a subset of 593 turns for signs of frustration including sighs, noticeable changes in intonation and lexical cues such as expletives. 11.5% of the turns were marked as frustrated and 30.8% of the dialogs had at least one frustrated turn.

We computed the understanding error rate for turns immediately following a non-understanding. While we will explore in more detail the different strategies used for reprompting in Section 4, Figure 1 gives the overall understanding error rate for those turns, in the presence or absence of each speaking style. As can be seen, all three styles are associated with an increase in understanding error rate (all differences are statistically significant at $p < 0.05$). This is in partial agreement with [8], who found in a study of the DARPA Communicator corpus that frustration was correlated with higher WER, but did not find such a correlation for hyper-articulation. Note that correlation does not mean that these speaking styles are causing the degradation in understanding (or WER), since, conversely, poor understanding is also likely to cause frustration, hyper-articulation, or loud speech. A controlled experiment remains to be done to tease apart these interfering effects.

4. Recovery Strategies: A Case Study

4.1. Default Non-understanding Strategy

The mismatches between user expectations and/or language described above lead to misrecognitions that cannot be solved by simply asking the user to repeat what they said. Rather, we explored two strategies: one consisted of giving examples of stops covered by the system, the other in asking the user which neighborhood they were leaving from, before refining to the exact stop name.

By default, the system uses a simple hand-crafted policy when faced with non-understandings. On the first non-understanding of a call, the system issues a prompt that notifies the user of the problem and explicitly recommends going to a quiet place, since we found that background noise was a major cause of recognition errors.

The user is also given examples of expressions that the system expects them to use at that point in the dialog (e.g. typical place names, times, or schedule navigation commands). On subsequent non-understandings, the user is always given examples, along with more or less help. The most verbose prompt includes a list of all the pieces of information that the system got from the user so far and an explanatory version of the current question.

Figure 2 shows the rate at which each concept was successfully acquired from the user. Note that users were not asked for their destination or travel time if the system detected that they were looking for uncovered bus routes or neighborhoods. Also, since the dialogs progress sequentially through the concepts in the order departure/bus route, destination, and time, many of the failed dialogs do not reach the point where the system asks about destination or time. Indeed, in the vast majority of the dialogues in which the departure place was acquired succesfully, the following concepts (destination and time) were also acquired.

4.2. User Responses to Non-Understandings

To better understand what happens when the user is faced with a non-understanding, we defined a labeling scheme based on the one proposed by Shin et al [9] to describe user responses to system prompts. In our case, we limited the labels to be of four kinds: **REP**, when the user exactly repeats their previous utterance, **RPH**, when the user rephrases their previous utterance while keeping the same semantics, **NEW**, when the user changes what they ask for, either by themselves or following a change of question from the system. The last class was **OTH** for all turns that would not fit in the previous three classes, including non-speech turns. Two of the authors labeled 2532 and 1120 utterances, with an agreement of 83% and a kappa of 0.65. In this study, we use the labels from the first labeler only on turns following a non-understanding, which amounts to 703 turns. The distribution across the three main classes is given in Figure 3, along with the understanding errors that each type of response triggered. First, it appears that all three types of responses were roughly equally frequent. Second, the **NEW** class yields significantly fewer understanding errors than the other two. This result is in agreement with what we have found in a different dialog system (see [10]). It also matches what Skanze [11] found in a Wizard-of-Oz situation, namely that, when unsure of what the user said, human wizards would predominantly ignore the utterance and try to pursue the task through a different path and ask different questions, rather than asking the user to repeat or rephrase. Also, although repeats and rephrases generate the same rate of understanding errors, the breakdown in misunderstand-

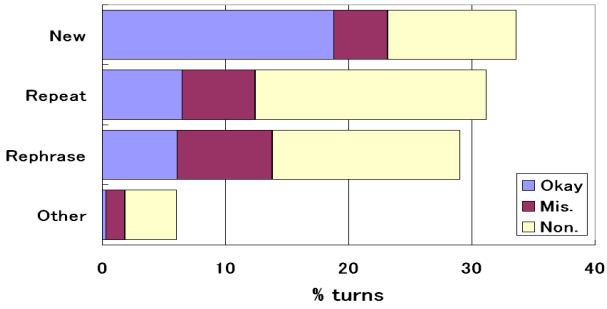

Figure 3: *Distribution of Understanding Errors by User Response Type*

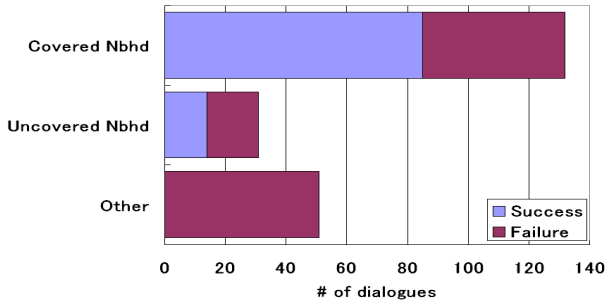

Figure 4: *Distribution of Neighborhood Backoff Strategy Success for Different User Request Types*

ings and non-understandings is different: repeats are more likely to trigger non-understandings (with $p < 0.05$) and rephrases are more likely to trigger misunderstandings (with $p < 0.1$). This should not come as a surprise since in the repeat case, users are repeating an utterance which already triggered a non-understanding. Nevertheless, dialog system designers should take this aspect into account when balancing the trade-off between non-understandings and misunderstandings, which typically have different costs and impact on the dialog.

4.3. Backing Off to Neighborhood Information

As seen in the last section, merely reprompting the user after a non-understanding is not always the right approach. Goldberg and her colleagues reported in [8] that simply repeating the system prompt leads to more user frustration than rephrasing it. They also found that the more times the user gets reprompted for a particular concept, the worse the WER and thus the understanding ratio. In order to avoid such understanding error spirals, we implemented a backoff strategy to obtain the departure place from the user. After two consecutive non-understanding at the beginning of the dialog, the system stops asking about the specific departure stop of the user and instead asks "*Which neighborhood do you want to leave from?*". If there are more non-understandings, examples of neighborhoods are given to the user. When a covered neighborhood is given and confirmed by the user, the system asks for the specific stop in the neighborhood. The user is also given the possibility to say "*I don't know*" to this last question, in which case the system uses a default stop for the neighborhood. When an uncovered neighborhood is given, the system informs the user that it currently does not support buses serving that neighborhood. Thus, this strategy has two goals: limiting the repetitiveness of the interaction in the presence of errors, and identifying out-of-coverage requests, which result in out-of-vocabulary words.

Figure 4 shows the distribution of calls where the backoff strategy was used, split by whether the user was asking for a neighborhood covered by the system, one that was in the PAAC database but not covered by the system, and neither of those two, i.e. a neighborhood not served by the PAAC buses or no neighborhood information at all.

5. Future Directions and Conclusion

We are continuing to support the public system and there are several improvements we would like to bring to this baseline system in the near future, along with more in-depth research questions we would like to address. First, we are going to use the collected data to improve recognition accuracy. We plan to investigate the use of acoustic models trained on frustrated or hyper-articulated speech. Another direction for research is the study of alternative strategies like the "backoff to neighborhood" introduced here. A controlled experiment would allow us to identify the strengths and weaknesses of this strategy as opposed to a system using only standard reprompting strategies.

6. Acknowledgments

This work is supported by the US National Science Foundation under grant number 0208835, "LET'S GO: improved speech interfaces for the general public". Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation. Thank you to the Port Authority of Allegheny County for access to their database and for their help in making the Let's Go system accessible to Pittsburghers.

7. References

[1] A. Raux, B. Langner, A. Black, and M. Eskenazi, "LET'S GO: Improving spoken dialog systems for the elderly and non-native," in *Eurospeech03*, Geneva, Switzerland, 2003.

[2] B. Langner and A. Black, "An examination of speech in noise and its effect on understandability for natural and synthetic speech," Language Technologies Institute, CMU, Pittsburgh PA, Tech. Rep. CMU-LTI-04-187, 2004.

[3] A. Raux and R. Singh, "Maximum likelihood adaptation of semi-continuous hmms by latent variable decomposition of state distributions," in *ICSLP 2004*, Jeju, Korea, 2004.

[4] A. Rudnicky, C. Bennett, A. Black, A. Chotimongkol, K. Lenzo, A. Oh, and R. Singh, "Task and domain specific modelling in the Carnegie Mellon Communicator system," in *ICSLP2000*, vol. II, Beijing, China., 2000, pp. 130–133.

[5] D. Bohus and A. Rudnicky, "RavenClaw: Dialog management using hierarchical task decomposition and an expectation agenda," in *Eurospeech03*, Geneva, Switzerland, 2003.

[6] A. Black and K. Lenzo, "Limited domain synthesis," in *ICSLP2000*, vol. II, Beijing, China., 2000, pp. 411–414.

[7] A. Raux, "Automated lexical adaptation and speaker clustering based on pronunciation habits for non-native speech recognition," in *ICSLP 2004*, Jeju Island, Korea, 2004.

[8] J. Goldberg, M. Ostendorf, and K. Kirchhoff, "The impact of response wording in error correction subdialogs," in *ISCA Workshop on Error Handling in Spoken Dialog Systems*, Chateau d'Oex, Vaud, Switzerland, 2003.

[9] J. Shin, S. Narayanan, L. Gerber, A. Kazemzadeh, and D. Byrd, "Analysis of user behavior under error conditions in spoken dialogs," in *ICSLP2002*, Denver, Colorado, 2002.

[10] D. Bohus and A. Rudnicky, "An empirical analysis of non-understanding and recovery strategies in spoken dilague systems," *submitted to SIGdial 2005*, Lisbon, Portugal, 2005.

[11] G. Skantze, "Exploring human error recovery strategies: Implications for spoken dialogue systems," *Speech Communication*, vol. 45, no. 3, 2005.