Learning of stochastic dialog models through a dialog simulation technique

Francisco Torres, Emilio Sanchis and Encarna Segarra

Departamento de Sistemas Informáticos y Computación
Universidad Politécnica de Valencia, Spain
{ftgoterr, esanchis, esegarra}@dsic.upv.es

Abstract

We present an approach for the learning of stochastic dialog models using a technique of automatic generation of dialogs. We have applied it to achieve a better performance in our dialog system, which answers telephone queries about train timetables in Spanish. Besides interacting with real users, the stochastic dialog manager can now interact with other module, in the role of the user, developing a large number of dialogs at a very low cost. From this interaction, the dialog manager is able to dynamically adapt its stochastic model, adding new transitions or modifying their probabilities, when a simulation ends satisfactorily. We expect that the modified model provides the dialog manager with a better strategy for answering real users than the strategy given by the initial model estimated from real dialogs.

1. Introduction

The development of spoken dialog systems based on statistical methods has reached interesting and well-founded results, as in [1], [2] and [3]. However, a serious problem in the refinement of these systems is the high cost of an evaluation made by interacting with real users. In consequence, in order to solve this bottleneck and provide an alternative way of improvement, several proposals have been presented in the last years. Different approaches about the use of simulated dialogs for learning the models and evaluating the systems can be found in [4], [5] and [6].

In [1] and [4], the need of a user simulator is justified by the number of iterations required for automatic learning of the optimal dialog strategy. The simulation is especially adequate in the early stages of this learning because real users would reject the initial, and too deficient, strategy. The simulated user is a stochastic generative model, obtained by supervised learning from a corpus. The probability distributions have been computed taking into account the last system turn and the current user state. Besides, the user modeling includes some pragmatic conventions and an intentional level of interaction.

In [5], a user simulator module is proposed as a tool that allows the refinement of the dialog system before real users can have access to it. The design of the user simulator requires the building of a corpus of dialog scenarios (that defines the aims of the users), and a corpus of sentences (recorded by several speakers, and adequate for dealing with the scenarios). Then, it is applied to test several confirmation strategies.

In [6], a method for simulating mixed initiative dialogs is presented. The user simulator has been stochastically modeled from a set of real user turns (obtained using a prototype of the system and collected in a corpus). Besides, in the specification of the user model, it has been supposed that the user behavior is goal directed. The technique has been applied to compare dialog strategies and to automatic learning of the strategies.

In this work, we present an approach for the learning of stochastic dialog models using a methodology similar to the ones briefly described above. Our research has been done in the framework of the BASURDE and DIHANA [7] projects, that deal with the development of a spoken dialog system to access an information system for train timetables. We have developed a stochastic dialog manager driven by semantics [8] and evaluated the dialog system [9]. Now, to improve its performance, by means of dynamic adaptation of its stochastic model, we have developed a user simulator. This module has been integrated in the dialog system to learn the dialog model.

2. Dialog system overview

In our dialog manager, we use stochastic models to establish the system strategy and semantic representations (frames) to describe the user dialog acts. In order to decide its response during the dialog, the dialog manager consults and updates two components: the stochastic dialog model (DM) and the historic register (HR). Figure 1 shows the algorithm.

```
Initialization(HR);  // Historic Register
Read(DM);  // Stochastic Dialog Model
DM.st = Opening;  // DM.st = state of DM
REPEAT
Read(U-frames);  // reading user frames
DM.input = Adapt(HR, U-frames);  // semantic generalization
DM.st = Transit(DM.st, DM.input);  // transition to user-state
HR = Update(HR, U-frames);  // HR update by user turn
DM.st = Transit(DM.st, HR);  // transition to system-state
HR = Update(HR, DM.st, BD.info);  // HR update by system turn
S-frames = Adapt(DM.st, HR);  // building system frames
Write(S-frames);  // writing system frames
UNTIL DM.st = Closing  // Closing, final state
```

Figure 1: Dialog manager algorithm

The DM, learnt from the BASURDE labeled dialog corpus (which contains 227 dialogs), is a bigram model. For each user (or system) dialog act (current state), it establishes the possible transitions to system (or user) dialog acts (next states), according to the probabilities of these transitions in the corpus.

The HR is a table that stores the values of the attributes, given by the user throughout the dialog. It also contains additional information about its state (associated confidences, confirmation state, updating time, etc.).
In our approach, the system performance suffers from certain lack of robustness of the dialog model, derived from the scarcity of the training samples. Thus, when new users speak to our system, some situations, that belong to the task but that are unseen in the BASURDE corpus, could occur. In these cases, the model would not provide any transition and, if the dialog manager were fully stochastic, it would lock up.

In order to solve these situations appropriately, the dialog manager follows a hybrid strategy, partially stochastic corpus based, and partially fixed by rules. The main mechanisms to complement the stochastic model are the use of a process that we called “semantic generalization” and the use of consistence rules with the content of the HR.

The semantic generalization is a preprocessing of the user frames. It is a help in the search for transitions to user states, by relaxing progressively the coincidence criteria among the received frames and the next states of the model. It can be seen as a kind of smoothing.

The consistence rules help in the search for transitions to system states. When the dialog manager has to decide its reply to the user (by selecting a transition in the DM), it applies these rules and chooses one from the subset of the possible transitions that are coherent with the content of the HR.

Recently [9], a process of error handling has been incorporated to the dialog system by means of the use of confidence measures in the dialog manager. We have evaluated this dialog system experimentally. The results show the system robustness: high success rates are reached (99% and 69%) even when the received user frames have a significant error rate (20% and 30%, respectively). The price is just a slight increase of the duration of the dialogs due to the necessary confirmations of the uncertain attributes.

Currently, a new and larger corpus (900 dialogs) has been acquired in the DIHANA project. We expect that the models trained from the DIHANA corpus will provide a better coverage of the task event space. Thus, using these models, the dialog manager will be able to follow a more stochastic, and less dependent on other complementary procedures, strategy.

3. Dialog system with user simulation

When our dialog manager will have to reply to the real users, (using models learnt either from BASURDE corpus or, soon, from DIHANA corpus), there will some probability of being in dialog situations that, although they were possible in the task, they not occurred in the corpus. Thus, the models, especially the BASURDE model, will not provide an adequate treatment of these situations.

As an alternative to the learning of the models from a corpus acquired interacting with real users, we have considered the learning by means of the automatic generation of dialogs. In this case, the acquisition is made interacting with user simulator modules. If this technique is applied, a simulated dialog corpus can be generated quickly and at a low cost. Besides, if the real user behavior is adequately modeled by the simulator modules, this synthetic corpus can be used to adapt the initial models (learned from the corpus of natural dialogs).

We have changed the dialog manager allowing it to modify the model. That is, when the dialog manager is used for the synthetic acquisition, it could add the selected transitions to the model and change their probabilities, temporarily or permanently depending on the success of the simulated dialog. Thus, the model can be adapted dynamically (during the same process of the generation of the dialogs).

The user simulation consists of two modules: a user dialog manager (UDM) and a user reply generator (URG). The UDM is the symmetric module of the system dialog manager (SDM). It receives the system frames, reads and writes transitions in the same model used by the SDM, applies a rule set to choose the transitions according to their aims, and generates the user frames. The URG is the symmetric module of the system reply generator (SRG). It receives the user frames and generates the corresponding natural language sentences.

Figure 2 shows the block diagram of the system, extended with the user simulator modules. Figure 3 shows the UDM, or user simulator, algorithm and the new SDM algorithm (a modified version of the previous one).

At the beginning of each dialog, the user simulator reads the parameters and objectives of the simulated scenario and stores this information in the UHR. Then, it reads the static version of the model (sDM), looks for a state of question of, at least, one objective, and generates the corresponding frame.

In each dialog turn, the user simulator reads the system frames, compares these frames and the possible system dialog acts, transits to a new system state in the sDM, updates the
UHR with data given by the system, transits to a new user state in the sDM, and generates the user frames. At the end of the dialog, the user simulator writes the UHR because the SDM needs it to verify the success of the dialog.

### USER SIMULATOR ALGORITHM

```plaintext
USER SIMULATOR ALGORITHM

Initialization(UHR); // UHR = User Historic Register
Read(sDM); // sDM = static Dialog Model
dDM.st = Question; // sDM.st = state of sDM
S-frames = Adapt(sDM.st, UHR); // user frames of initial state
Write(U-frames);

REPEAT
  Read(S-frames); // reading system frames
  sDM.input = Adapt(UHR, S-frames); // semantic generaliz.
  sDM.st = Transit(sDM.st, sDM.input); // transition to system-st
  UHR = Update(UHR, S-frames); // HR update by S turn
  sDM.st = Transit (sDM.st, UHR, Rules); // transit to user-st
  U-frames = Adapt(sDM.st, UHR); // building user frames
  Write(U-frames); // writing user frames
UNTIL sDM.st = Closing;
Write(UHR)
```

### NEW SYSTEM DIALOG MANAGER ALGORITHM

```plaintext
NEW SYSTEM DIALOG MANAGER ALGORITHM

Initialization(SHR); // SHR = System Historic Register
Read(dDM); // dDM = dynamic Dialog Model
dDM.st = Opening; // dDM.st = state of dDM
Repeat
  Read(U-frames); // reading user frames
  dDM.input = Adapt(SHR, U-frames); // semantic generaliz.
  dDM.st = Transit(dDM.st, dDM.input); // transition to user-st
  SHR = Update(SHR, U-frames); // HR update by U turn
  dDM.st = Transit(dDM.st, SHR); // transition to system-st
  SHR = Update(SHR, dDM.st, BD.info); // HR update by S turn
  S-frames = Adapt(dDM.state, SHR); // building system frames
  Write(S-frames); // writing system frames
UNTIL dDM.st = Closing;
Read(UHR);
If success = Compare(UHR, SHR);
IF success THEN Write(dDM, transitions)
```

### Figure 3: UDM and SDM algorithms

As it can be observed, the SDM and the UDM algorithms are very similar. The main difference between them is in the procedure of choosing selections in its own turn. The UDM processes the system turn, using DM as a bigram model and applying semantic generalization in the same way that SDM does it (lines 1 y 2 in the figure 3).

However, when the UDM decides its reply (in its own turn, or user turn), it uses the DM as a list of possible new states, without regard of the existence of transitions to them. Then, it applies a set of rules to search for the best new user state, according to the current content of the UHR and given a collaborative strategy, which looks for satisfying the aims of the simulated scenario (line 3 in the figure 3).

Figure 4 show the main rules for choosing a user state. The first rule establishes that, after a system answer, the UDM verifies the existence of objectives that are still unknown and, depending on this, it transits either to a question state or to the closing state.

The second rule establishes that, after a system question about unknown attributes, the UDM searches for the values of the requested attributes and, if it finds them (because the simulated scenario gives them values), it transits to an answer state. Otherwise, the transition is made to some question state of an objective attribute. In the last case, there is still some probability of satisfying part of the system question by providing some data spontaneously.

The third rule establishes that, after a request for attribute values confirmation, the UDM verifies that the values of the requested attributes are known, compares them with those hypothesized by the system and, depending on their equality, it transits either to a affirmation state or to a negation state. Again, in case of not knowing the requested attributes, it transits to a question state of an objective attribute.

### Figure 4: Rules applied by the UDM

<table>
<thead>
<tr>
<th>Condition</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>System state = Answer</td>
<td>THEN look for new targets in UHR</td>
</tr>
<tr>
<td>IF attrib / is-target &amp; unknown-value</td>
<td>THEN look-for &amp; transit-to user-st = Question of the attrib</td>
</tr>
<tr>
<td>ELSE look-for &amp; transit-to user-st = Closing</td>
<td></td>
</tr>
</tbody>
</table>

### Figure 5: Simulations success evaluation algorithm

With regard to the new SDM algorithm, the two main changes are the following. In each dialog turn, the selected transitions are temporarily added to the model, readjusting the probabilities in concordance. At the end of the dialog, the UHR is read, and the two registers (UHR, SHR) are compared in order to determine the success of the simulation. All the transitions that were temporarily added to the model, in case of successful ending, are permanently added and, otherwise, are removed, leaving the model as it was at the beginning of the dialog. Figure 5 shows the algorithm used for the automatic evaluation of the success of the simulations.

### 4. Evaluation of the models

We can report some results about the learning of the stochastic dialog models using the described dialog simulation technique. These results, although preliminary, point to the feasibility of this technique and so they encourage us to do further research.
We have defined 14 scenarios, which consist of queries about timetables, prices and/or train types for one-way trips. We have run 4000 simulations.

Figure 6 shows the evolution of the probabilities of some transitions during the experiments. We have chosen them because they illustrate some interesting changes. There are reasonable transitions (e.g., the transition from a system state confirming a destination or origin to a user state accepting a destination or origin) with significant probabilities in the new model, DM*, whereas they did not exist (i.e., they had zero probability) in the initial DM. We also have found other reasonable transitions that appear in both models, but with a higher probability in the DM* (e.g., the transition from a state system answering departure timetables to a user state asking for arrival timetables). On the other hand, there are transitions with lower probability in the DM* than in the DM.

It should also be noted, in figure 6, that, after running over 1000 simulations, the probabilities tend to converge. Therefore, we can conclude that the proposed technique allows the dynamic adaptation of the DM in a stable way, converging after a moderate number of iterations.

5. Conclusions

We have proposed a methodology that allows the learning of dialog strategies. It is possible to create new transitions (i.e. cases with zero probability in the DM, that correspond to unseen situations in the training corpus), to increase the probabilities of some reasonable transitions, and to reduce the probabilities of other transitions. We expect that the dialog manager achieves a better performance using the new DM*, updated after running the simulation experiments.

As future work, there are some obvious aspects to improve the user simulation. We have to extend the table of the simulated scenarios to all the scenarios defined for the DIHANA task. We have to incorporate recognition errors, either real or simulated, in order to broaden the exploration of the event space. Finally, we will apply the same technique to the initial model trained from DIHANA corpus.

6. Acknowledgements

This work has been partially funded by CICYT under project TIC2002-04103-C03-03, Spain.

7. References