OPTIMIZE THE OBVIOUS: AUTOMATIC CALL FLOW GENERATION

D. Suendermann, J. Liscombe, R. Pieraccini

SpeechCycle Labs, New York, USA

{david, jackson, roberto}@speechcycle.com

ABSTRACT

In commercial spoken dialog systems, call flows are built by call flow designers implementing a predefined business logic. While it may appear obvious from this logic how the call flow has to look like, i.e., which pieces of information have to be gathered from the caller or back-end systems and in which sequence, there are, in fact, strong arguments for automating call flow generation:

- manual generation is time-consuming
- manual generation is suboptimal and error-prone
- automatic generation can react on dynamically changing business logic or external factors such as the distribution of callers and call reasons

This paper presents a method for automatically deriving a call flow minimizing the average number of user turns given a business logic and a frequency distribution of call reasons. As an example, we applied the method to a call routing application whose manually built call flow is processing about 4 million calls per month and whose call reason distribution served to measure the impact of the automatic call flow generation.

Index Terms— automatic call flow generation, spoken dialog systems

1. INTRODUCTION

Commercial spoken dialog systems almost exclusively rely on the call flow paradigm, one of many possible dialog management strategies [1]. A call flow is a graph where the nodes represent dialog states and the arcs represent transitions conditioned on caller or back-end system input (see e.g. Figure 1). Generally, call flows are built by call flow designers (aka voice user interface designers or interaction designers) implementing a given business logic. This business logic can be provided in form of tables such as Table 1 which was basis of the aforementioned example call flow.

Call flows can be extremely complex with thousands of nodes and transitions, e.g. when representing a problem-solving spoken dialog application application [2]. These

Patent pending.

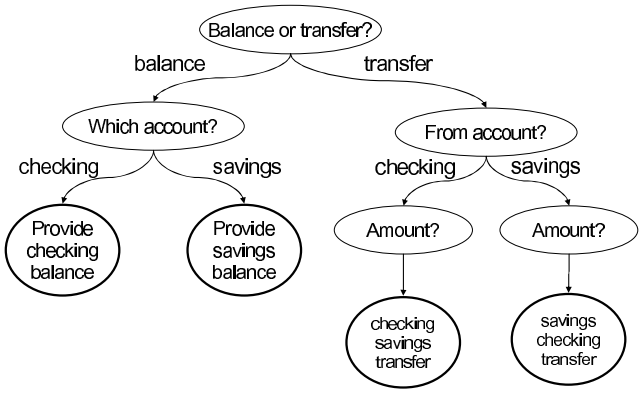

Fig. 1. Example of a call flow.

complex applications, however, can be broken down into sub-dialogs with their respective sub-call flows each of them serving specific purposes such as problem capture, equipment type collection, resolution steps, etc. For instance, the above given example can be part of a large phone banking application able to serve many purposes such as fund management, review of credit card statements, transfer to external accounts, etc.

Without loss of generality, in this paper, we consider a (sub-)call flow to be of a question-answer-destination type. That is, the business logic tables consist of a number of columns representing questions whose possible answers are listed in the column fields. The mandatory destination column determines the routing destination or final action performed by the call flow. A general business logic table looks as shown in Table 2. Here, P^n are the relative frequencies, or estimated probabilities, of how often the n^{th} table row will

service type?	account type?	amount?	destination
balance	checking		give checking balance
balance	savings		give savings balance
transfer	checking	x\$	check.-sav. transfer x\$
transfer	savings	x\$	sav.-check. transfer x\$

Table 1. Example of a business logic table.

A_1	A_2	...	A_M	D	P
A_1^1	A_2^1	...	A_M^1	D^1	P^1
A_1^2	A_2^2	...	A_M^2	D^2	P^2
$\vdots$	$\vdots$		$\vdots$	$\vdots$	$\vdots$
A_1^N	A_2^N	...	A_M^N	D^N	P^N

Table 2. General business logic table.

be visited. These probabilities are essential for some of the following sections' considerations. If they are not known as for instance during the projection phase of a call flow, they may very well be estimated or even set to unity to get started. It is also important to note that answers to questions can be retrieved by a variety of means including

- prompting the caller using a directed dialog listing all possible answers
- prompting the caller with an open prompt
- accessing back-end information
- caller-initiated system input

So, why would we try to come up with an automatism to derive a call flow from this table rather than asking Q_1 (the question whose answer is A_1) followed by Q_2 followed by Q_3, etc.? Mainly, because

Questions have different levels of relevance.

This includes the observation that questions may be completely irrelevant as manifested in the business logic table (like for instance the question about the transfer amount when calling for the account balance) or due to preexisting information gathered by back-end systems or caller-initiated input. Moreover, asking questions in order of decreasing relevance leads to shorter conversations. E.g., in Table 3, when we ask Q_1 first and get b as response, Q_2 can be skipped and the call can be directly routed to D^3. Similarly, if we would ask Q_2 first and get c as response, we would not have to ask Q_1 anymore but could directly route to D^1. So, which strategy is better? Imagine, most of the calls end up at destination D^1, then it is most reasonable to ask Q_2 first as the answer would most likely be c. So, we have to look at the probabilities P^n to come up with that strategy optimizing the relevance of questions and, hence, minimizing the average number of questions asked. Besides, to ask more relevant questions first and to rate relevance dynamically based on perceived probabilities is a very human approach: When we enter a coffee shop during winter it is unlikely that we get welcomed with the words "Would you like a lot of ice in your soda?"

The question of how to best order information-gathering questions in spoken dialog systems has been addressed before. Some researchers (e.g. [3], [4]) have proposed user models whereby the importance of information is determined by storing known preferences of a user and using this to order presentation elements for future interactions with that user. However, this approach is not relevant for an application, such as the once discussed here, where the number of repeat callers is relatively low and the ability to collect preference information is not feasible. Another proposed approach is to learn question ordering from human-human dialogs. For example, [5] used a human-human corpus to train a decision tree that optimizes the order of questions such that information gain is maximized. In so doing, the dialog flow was also maximally shortened and it was furthermore reported that call satisfaction increased as well. In subsequent sections, we describe a similar method for optimizing question presentation order, but instead of relying on human-human interaction for learning—something that is not always possible or cost-effective to obtain—we optimize on the ultimate route points of millions of callers using spoken dialog systems in production.

A_1	A_2	D	P
a	c	D^1	P^1
a	d	D^2	P^2
b	d	D^3	P^3

Table 3. Example of a business logic table.

2. THE ALGORITHM

2.1. On relevance and information gain

As motivated above, we want to come up with a strategy to ask the questions in a call flow in decreasing order of relevance before routing to the destination or launching the final action D^n. Since a call flow as introduced above is similar to what in machine learning is referred to as *decision tree*, we can use well-established machine learning techniques to come up with an optimal call flow. When we agree that the most relevant question is that one whose answer provides us the maximum information, we can borrow the *information gain* measure as defined in [6] to get started:

$$I(Y;X) = H(X) + H(Y) - H(X,Y)\,. \qquad (1)$$

Here, X is a given attribute, i.e. the answer to a question, and Y is a class attribute, i.e. the sought-for destination. So, we may as well write

$$I(D;A_m) = H(A_m) + H(D) - H(A_m,D)\,. \qquad (2)$$

H is Shannon's entropy [7] defined as, e.g.

$$H(D) = -\sum_{\delta=1}^{\Delta} P(\delta)\log_2 P(\delta) \qquad (3)$$

with $\delta \in \{1, \ldots, \Delta\}$ being the distinct destinations in the currently processed business logic table. So, at every node in the call flow, we simply determine which question leads to the maximum performance gain:

$$Q_{\hat{m}} \text{ with } \hat{m} = \underset{m=1,\ldots,M}{\arg\max} I(D; A_m) \,. \tag{4}$$

If none of the questions leads to a performance gain, there is nothing left to do, and we can route to the final destination. As an example, consider that $D^1 = D^2 = D^3$ in Table 3. As we have a single destination, we get $H(D) = -1 \cdot log_2(1) = 0$. Furthermore, we know that $P(\alpha_m, \delta) = P(\alpha_m)$ (α_m being the distinct answers of A_m) since the entire probability mass is associated with the single destination δ. This leads to $H(A_m, \delta) = H(A_m)$ and, hence, to $I(D; A_m) = 0$. So, we are good to route in this scenario.

2.2. Step by step

The algorithm $f(T)$ providing a call flow from a given business logic table T works iteratively as follows:

- Compute the information gain values $I(D; A_m)$ according to Equation 2.
- If $I(D; A_m) = 0$ for $m \in \{1, \ldots, M\}$, route to D. If D is not unique, the original business logic table was inconsistent (contradicting rows) and needs to be fixed. Exit $f(T)$.
- Determine the optimum question $Q_{\hat{m}}$ according to Equation 4.
- Iterate over all possible answers $\alpha_{\hat{m}}$
  - setting $T' := T$,
  - eliminating all rows but those associated with $\alpha_{\hat{m}}$ in T',
  - eliminating the $\hat{m}$'s question column in T',
  - calling $f(T')$.

2.3. Special properties

Section 2.2 shows a raw version of the call flow deduction algorithm that, in practical applications, can be enriched by a number of special features and constraints, some of which are to be listed here:

- The use of open prompts, user-initiated input, backend integration, or other input modalities may provide information prior or during the conversation. Therefore, call flows should generally be computed during runtime. If possible, $f(T')$ for the next machine-user interaction should be called right before the interaction is carried out taking all available information into account. This way, also re-gathering of information is avoided as in the following typical example:

  A: "You can get your balance or make a transfer. Which one would you like?"

  C: "The balance of my checking account, please."

  A: "Would you like to hear the checking or savings account balance?"

- There can be special fields or special columns in the business logic table such as
  - Wildcard fields (matching everything) and negation fields (matching everything but its content). These are to keep the business logic table as short as possible.
  - Value fields. Conventional fields represent a single possible answer (such as the *balance* field in Table 1). In contrast, value fields can represent a variety of inputs (such as a numerical value, a set of responses, or even a whole sub-call flow). The integration of their respective information gain contribution is a little more elaborate and beyond the scope of this paper.
  - Priority columns/column types. Although sometimes a certain question may produce the highest information gain, it may be unnatural to ask before another question has been answered. E.g., a music instruments hotline will probably not talk like this:

    A: "Welcome to Mewtheex. Are you calling about a red, blue, or black instrument?"

    C: "Uuh. I don't care."

    A: "Do you need repair or do you want to buy one?"

    C: "Buying, I guess."

    A: "Do you want to pay by credit card or check?"

    C: "Uuh?!"

    A: "And... which instrument is it about: ukulele, piccolo, or triangle. You can also say: *Give me a different instrument.*"

    C: "What the hell?! I need an Eliminator Demon Drive Double Bass Drum Pedal!"

    The resolution for this scenario is to introduce columns of different priorities or types whose dependence is taken into consideration when computing the optimal information gain.

- There can be additional constraints necessary to have the algorithm produce call flows following certain business requirements or voice user interface best practices such as limiting the number of possible choices at a given directed dialog to a certain maximum [8].

number of calls	$3,868,014$
number of questions	$M = 4$
number of rows	$N = 31$
number of distinct destinations	$\Delta = 20$
average number of call flow questions	$\hat{M} = 2.87$

Table 4. Experiment parameters.

3. AN EXAMPLE

In order to test the proposed algorithm in a real-world scenario, we took the business logic table from a mature call router application of a large cable company processing about 4 million calls per month (see [9] for details). Based on call logs collected over a one-month time frame (August 2009), the probabilities P^n were estimated, and a static call flow was generated based on the algorithm described in Section 2.2. Table 4 reports on the experiment's parameters.

Since P^n was known, we were able to estimate the probability with which each transition of the generated call flow was taken and, hence, were able to estimate the average number of questions asked to route a caller to one of the 20 final destinations. We obtained an average number of $\hat{M} = 2.87$ questions for the given business logic. The original business logic was based on four questions:

- service type (orders, billing, technical support, etc.)
- product (internet, cable TV, telephone)
- actions (cancel, schedule, make a payment, etc.)
- modifiers (credit card, pay-per-view, digital TV conversion, etc.)

Hence, the implementation of said business logic one-question-after-the-other[1] as discussed in Section 1 would have resulted in an average of four questions. This value could be reduced by almost 30% using automatic call flow generation.

4. CONCLUSION

Considering the average duration of a question-response interaction together with common per-minute fees for interactive voice response hosting and the aforementioned monthly call volume of the application, a 30% reduction of the average number of asked questions due to automatic call flow generation could translate to savings of five- to six-figure US dollar amounts per month. Furthermore, the likelihood of callers opting out (i.e., asking for an agent) or hanging up out of frustration decreases as the conversation speeds up leading to higher resolution rates and better caller experience. Finally, automatic call flow optimization as compared to the conventional anecdotic and ad-hoc tuning of the call flow logic can be magnitudes faster and provide functionality manual call flows cannot use such as dynamic call flow generation based on a variety of runtime inputs from user and back-end systems.

[1] At this point, it should be mentioned that the actual production system underwent a continuous tuning cycle to manually optimize the voice user interface. Several instances of open prompts and user-initiated input were implemented to cut down the number of questions to a maximum of 3. However, this effort took several voice interaction designers and speech scientists more than one year of careful analysis and implementation work.

5. ACKNOWLEDGEMENTS

Some of this work's ideas saw the light of day in brainstorming sessions involving the following of our dear colleagues: Krishna Dayanidhi, Peter Krogh, Michael Levy, James Mesbur, and Eric Woudenberg.

6. REFERENCES

[1] W. Minker and S. Bennacef, *Speech and Human-Machine Dialog*, Springer, New York, USA, 2004.

[2] K. Acomb, J. Bloom, K. Dayanidhi, P. Hunter, P. Krogh, E. Levin, and R. Pieraccini, "Technical Support Dialog Systems: Issues, Problems, and Solutions," in *Proc. of the HLT-NAACL*, Rochester, USA, 2007.

[3] M. Walker, S. Whittaker, A. Stent, P. Maloor, J. Moore, M. Johnston, and G. Vasireddy, "Generation and Evaluation of User Tailored Responses in Dialogue," *Cognitive Science*, vol. 28, no. 5, 2004.

[4] V. Demberg and J. Moore, "Information Presentation in Spoken Dialog Systems," in *Proc. of the EACL*, Trento, Italy, 2006.

[5] P. Fodor, "Dialog Management for Decision Processes," in *Proc. of the 3rd Language and Technology Conference: Human Language Technologies as a Challenge for Computer Science and Linguistics*, Poznan, Poland, 2007.

[6] T. Mitchell, *Machine Learning*, McGraw Hill, New York, USA, 1997.

[7] C. Shannon, "A Mathematical Theory of Communication," *Bell System Technical Journal*, vol. 27, 1948.

[8] M. Cohen, J. Giangola, and J. Balogh, *Voice User Interface Design*, Addison Wesley, Redwood City, USA, 2004.

[9] D. Suendermann, J. Liscombe, K. Evanini, K. Dayanidhi, and R. Pieraccini, "From Rule-Based to Statistical Grammars: Continuous Improvement of Large-Scale Spoken Dialog Systems," in *Proc. of the ICASSP*, Taipei, Taiwan, 2009.