Minimally Invasive Surgery for Spoken Dialog Systems

David Suendermann, Jackson Liscombe, Roberto Pieraccini

SpeechCycle Labs, New York, USA

{david, jackson, roberto}@speechcycle.com

Excerpt

We demonstrate three techniques (Escalator, Engager, and EverywhereContender) designed to optimize performance of commercial spoken dialog systems. These techniques have in common that they produce very small or no negative performance impact even during a potential experimental phase. This is because they can either be applied offline to data collected on a deployed system, or they can be incorporated conservatively such that only a low percentage of calls will get affected until the optimal strategy becomes apparent.

Index Terms: spoken dialog systems, performance optimization, minimal invasion

1. Entrée

The main argument for using commercial spoken dialog systems is to replace the human agent role in a telephone conversation in order to save costs [1]. Other arguments such as consistency of performance or ease of scalability can also be mapped to cost savings. This is since call center agents can be extensively, consistently, and persistently trained (which is expensive), and a good number of agents could be kept on-call to account for unexpected peek situations like during outages (which is also expensive).

When spoken dialog systems provide such multi-fashioned savings, can we quantify them? And if so, what can we do to optimize these savings?

Every successfully automated call prevented a human agent to handle the same call, so, there is a (potentially call-type-dependent) saving amount associated with this call. This amount W_A can be estimated based on statistics drawn from call center transactions. On the other hand, automated calls produce costs such as hosting, licensing, or telephony fees which depend on the duration of the call T. The per-time-unit cost W_T can be calculated considering the former (and other) factors. Formally, the savings for a call are

$$S = W_A A - W_T T \; . \tag{1}$$

Here, the flag $A \in \{0, 1\}$ determines whether the call was automated or not. For the sake of simplicity, we regard W_A and W_T as call-independent constants in the following, so, for a set of calls $1, \ldots, N$ with the respective automation flags $A_1, \ldots, A_N$ and the durations $T_1, \ldots, T_N$, we can estimate the average savings as

$$\bar{S} = \frac{1}{N} \sum_{n=1}^{N} W_A A_n - W_T T_n \; . \tag{2}$$

Any optimization technique we implement may have an impact on the individual calls' automation flags or durations, so, in fact, they are variables depending on the system in use. Without loss of generality, the system in use can be described by a system parameter vector ξ describing one particular system out of the set of all possible systems Ξ. Now, to optimize a spoken dialog system is to use that parameter vector $\hat{\xi} \in \Xi$ yielding the maximum average savings

$$\begin{aligned} \hat{\xi} &= \arg\max_{\xi \in \Xi} \frac{1}{N} \sum_{n=1}^{N} W_A A_n(\xi) - W_T T_n(\xi) \\ &= \arg\max_{\xi \in \Xi} \sum_{n=1}^{N} T_A A_n(\xi) - T_n(\xi) \end{aligned} \tag{3}$$

with $T_A = \frac{W_A}{W_T}$, a parameter describing the trade-off between savings induced by automation and costs induced by duration. Its unit is in time domain, and it can be interpreted as the duration of an automated call for which savings and costs are en par. As example, let us consider that a human operator costs 12.50 US$ for successfully handling a certain type of call. Let us further assume the software-as-a-service vendor of the spoken dialog system charges 15 US cents per minute for an automated call of the same type. We calculate $T_A = 5,000$s for this scenario.

Returning to the idea of a reward function, Equation 3 demonstrates that the reward can also be expressed in terms of agent time saved by the application:

$$R = T_A A - T. \tag{4}$$

In contrast to Equation 1, this representation avoids speaking in currency units, unbecoming for scientific publications in the authors' opinion.

This paper focuses on the optimization of commercial spoken dialog systems based on the above promoted notion of reward function. The fact that we primarily look at commercial systems severely constrains the way such optimization can be achieved. This is because the stability of customer-facing production systems is first priority. Systems taking traffic must be

guaranteed to perform at a certain minimum performance level. Any applied changes should be guaranteed to boost or only minimally hurt performance.

To that effect, two of the optimization strategies covered below (Escalator in Section 2 and Engager in Section 3) are applied offline to a formerly recorded set of calls. This is possible, as the impact these strategies would have had if they would have taken the same life calls can be predicted based on certain assumptions discussed in the respective sections. In contrast, EverywhereContender (see Section 4) can indeed affect automation of calls. However, the negative impact this may potentially cause can be minimized by conservatively adjusting parameters as described in the respective section.

Having set the stage for minimally invasive surgery techniques, we would like to briefly return to the above introduced way of making optimal decisions among competing systems. Equation 3 is mathematically somewhat imprecise in that

1. There may not be equally many samples (N) drawn for each system ξ.
2. The average savings for a system ξ may not be statistically stable enough to make a hard decision on what the best system is.

Escalator and Engager are both based on offline simulation on formerly collected data and therefore come along with a constant N for all involved comparisons. EverywhereContender is an online technique that relies on splitting traffic among multiple systems with strongly varying degrees, so, N highly dependends on the system ξ. Moreover, it is possible that $N(\xi)$ is very small such that the arg max operation of Equation 3 becomes unreliable. Here, hard decisions have to be replaced by probability-based soft decisions as done in Section 4.

2. Escalator

Looking at Equation 1, we see that non-automated calls always result in negative savings proportional to the handling time of the call. Consequently, reducing durations of non-automated calls would lead to an increase of average savings. While this is true for both automated as well as non-automated calls, non-automated ones can be shortened aggressively by escalating to an agent as early as possible. Optimally, if there were an oracle telling us at the beginning of a call whether the call will end up automated or not, one could escalate non-automated calls before even entering the application.

We call an algorithm that deliberately escalates calls based on its opinion about the call outcome *Escalator*. While the concept of an Escalator (especially when talking about oracles) may sound like an advanced topic, it does not have to be very sophisticated. In fact, every spoken dialog system that transfers callers to agents in a given situation already contains an instance of an Escalator. Here, escalation reasons may simply include unsolicited agent requests, speech recognition and understanding problems, or situations the spoken dialog system does not know how to handle. More sophisticated Escalators estimate the probability of the current call ending unsuccessfully at certain time points throughout the call and escalate if a maximum probability threshold is exceeded [2, 3]. The probability estimator can be based on any sort of features found to be of predictive power. These features can include the whole dialog history up to the current moment including transitions taken, textual and acoustic speech input, respective acoustic and semantic confidence scores, backend information, number and speed of interaction turns, number of no-match, no-input, or dis-confirmation events, &c.

Hence, the Escalator's probability estimation may involve a rather complex algorithm requiring data input streams not available in many commercial spoken dialog architectures. E.g., the common architecture of a dialog system uses speech data only at the front-end. The recognizer transforms the acoustic input into a word string or graph, and the language understanding component extracts semantic contents from the recognizer's output. The Escalator, however, is part of the dialog manager that only sees the output of the language understanding component but does not have visibility to the acoustic features anymore.

A more straight-forward implementation of an Escalator that does not require any type of feature processing at run-time is to prune the call flow removing nodes and paths proven to negatively impact the system's overall performance. One way to do this is to compute a node-dependent performance score by calculating the average reward of calls hitting a specific node. This can be done by evaluating log data of calls processed by the application in question. For every node in the application, this average score is computed resulting in a ranking list of node-wise performances. Now, we want to see how the application's overall performance changes by incrementally eliminating nodes (and all paths coming from this node) starting from the worst performing node.

Luckily, call flow pruning can be simulated in an offline experiment (satisfying our demand for minimal invasion). This is because we are able to tell the reward score a call would have produced if it would have been escalated at a certain node: We know the call duration at the moment of early escalation, and we know the fact whether the call was considered automated at this moment or not[1].

In order to demonstrate the effectiveness of the pruning Escalator, we used log data taken from a cable Internet troubleshooting application collected over a period of four days in March 2010. Table 1 contains the corpus statistics of this experiment.

Table 1: *Corpus statistics Escalator experiment*

#calls (tokens)	45,631
#nodes (types)	847
#nodes pruned	176
T_A	5,000s
R w/o pruning	183.5s
R w/ pruning	196.8s
ΔR	13.3s

Figure 1 shows how the reward as defined in Equation 4 evolves with more and more nodes pruned. An optimum is reached for 176 pruned node types.

[1] In almost all cases, an early escalated call will be non-automated and, consequently, result in a negative reward. The effectiveness of call flow pruning comes from the fact that there may be many nodes and paths whose individual automation rate is extremely low. The handling time may have a worse effect on the average reward than the few automated calls can compensate for.

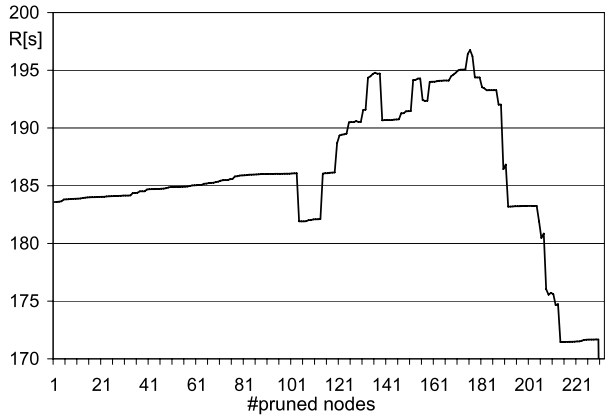

Figure 1: *Optimizing an Escalator based on call flow pruning*

3. Engager

The major weakness of the Escalator technique is that it has a potential negative effect on the automation rate. The overall reward gain is only achieved by reducing handling time much more effectively than negatively affecting automation rate. When pruning gets too aggressive (in the above example for more than 176 node types) then the reduction of the automation rate dominates the game, and the reward curve starts falling.

Can we possibly reduce handling time without negatively impacting automation? While there are multiple straightforward ways to do so (such as speeding up prompts, reducing latency caused by technical issues as slow network, underperforming backend databases, or sub-optimal file caching), there is also a less obvious technique we baptized Engager because it engages the current application design in a complete resynthesis.

One of the questions coming up during the design of a spoken dialog application is the optimal sequence with which the different nodes are to follow each other. Imagine we want to find out which modem a caller is using out of a variety of three: a black Motorola, a white Motorola, and a black Scientific Atlanta. Now, we may not want to ask for the two dimensions color and brand in one question to avoid confusion and therefore split the disambiguation task in two possible questions:

A Is your modem black or white?

B Do you have a Motorola or a Scientific Atlanta modem?

Naturally, if the caller responds "white" to A, it is a Motorola, so we do not have to ask B. However, we could also ask B first, and when the response is "Scientific Atlanta", the color is black, and we would not have to ask A anymore. So, which one is better A→B or B→A?

The optimal order is that one that minimizes the average number of asked questions per call, hence, resulting in the lowest average handling time. Say, we know that 50% of the callers have a Motorola and 80% have a black modem. The order A→B would then have us ask the second question in 80% of the calls, so, the average number of questions per call would be 1.8. B→A would result in only 1.5 questions asked in average and would therefore be the preferred order in this scenario.

At the design time of an application, such statistics may not be available hence forcing the interaction designer to make arbitrary decisions on the orders of nodes. However, once the applications starts taking traffic, statistics become available, and nodes can be re-ordered to optimize for handling time.

In [4], we applied Engager to a call routing application. The experiment's properties are shown in Table 2. After collecting statistics from almost 4 million calls, optimal reordering of the questions resulted in a cut of 1.13 questions per call on average or an average reward gain of 10.5s.

Table 2: *Corpus statistics Engager experiment*

#calls (tokens)	3,868,014
#routing points	20
#questions w/o Engager	4
#questions w/ Engager	2.87
ΔR	10.5s

4. EverywhereContender

Pruning and reordering nodes are apparently effective strategies to optimize spoken dialog systems. However, both of them achieve a reward gain mainly by reducing an application's handling time. Is there no way, we can positively influence automation rate, the other addend to the reward function?

In fact, looking at a given application, there may be a thousand things one can think of changing with a potential impact on automation rate. Is directed dialog best in this context? Or open prompt? Open prompt given an example? Or two? Or open prompt but offering a backup menu? Or a yes/no question followed by an open prompt when the caller says no? What are the best examples? How much time should I wait before I offer the backup menu? Which is the ideal confirmation threshold? What about the voice activity detection sensitivity? When should I time out? What is the best strategy following a no-match? Touch-tone in the first or only in the second no-match prompt? Or should I go directly to the backup menu after a no-match? What in the case of a time-out? &c.

Not even the smartest interaction designer will be able to come up with an optimal decision. So, can we do something similar like we did with Escalator and Engager? Can we look at data to find out what is optimal? When we allow for a litte degree of live experimentation, we can. We can implement all of the above solutions into our application and let them contend. We can route a certain amount of traffic to each of the contenders by randomly choosing one of them in every call and then measure the average reward for each contender by looking at data. The invasiveness of this approach can be reduced by lowering the amount of traffic hitting certain contenders expected to be underperforming.

Generally, the randomizer that decides which of the contender gets a given call, uses a set of weights associated with each of the contenders to decide how much traffic to route to which contender. When these weights are real numbers summing up to one, then each of them can be interpreted as the probability with which the respective contender is chosen. Ideally, it should be the probability that the respective contender is the winner, taking all available statistical information into account. E.g., when there were two contenders A and B, and we

were able to tell from data that A is the winner with a 80% chance, 80% of the traffic should be routed to A. While collecting more and more data, the probabilities keep changing, and the traffic hitting each contender keeps changing accordingly, until, at some point in time, a definitive winner is found.

The estimation of the contender probabilities can be based on statistical tests such as t and z tests [5, 6] for two-way contender splits. Here, the probability of a contender is the p value of observing a value for the test statistic that, assuming the null hypothesis being true, is at least as extreme as the value that was actually observed. Consequently, statistical significance of the contender approach is inherent to the probability estimation. A contender is significantly underperforming when its probability falls under, say, 1%, i.e., a p value of 0.01. In case of an n-way contender split, the numeric solution of n-dimensional integrals over the probability distributions of each of the contenders is required.

As an example, we applied three contender splits to a cable television troubleshooting application, one four-way split and two two-way splits. Table 3 contains the experimental properties. Since our experiment was based on less than 40,000 calls, with a single exception, none of the contenders was found to perform statistically significantly worse than its competitors. Only one contender of the four-way split resulted in a probability of less than 1%. Despite this lack of data to make final decisions, the adapted probabilities resulted in an overall performance gain of 29.4s compared to the baseline system which used equiprobable contenders. This demonstrates the effectiveness of the EverywhereContender approach in that it has the potential of positively impacting not only handling time but also automation rate.

Table 3: *Corpus statistics EverywhereContender experiment*

#calls (tokens)	38,004
T_A	5,000s
R baseline	253.4s
R after contending	282.9s
ΔR	29.4s

Similar to the two types of Escalators we discussed in Section 2, the one based on pure offline pruning (static) and the one based on a probability estimator during the course of a call (dynamic), there can be two ways of implementing the contender probability estimator. The probability weights can be calculated as described above and be static to a given contender split. On the other hand, the probability may depend on run-time variables such as the identity of the caller, the season, day of the week, or time of the say, the response to questions in the history of the call, or even acoustic parameters indicating a certain caller behavior. This type of dynamic contender model may even further drive system performance, however, as in the case of Escalator, it requires advanced intergration and control mechanisms and may come along with a higher degree of invasion and potential unpredictable behavior. In that, the contender approach resembles other approaches to statistical spoken dialog system design such as reinforcement learning [7], or partially observable Markov decision processes [8].

5. End

This paper discussed three techniques for optimizing performance of spoken dialog systems by doing either offline simulations (Escalator, Engager) or minimally invasive production experimentation (EverywhereContender). The impact of these techniques was shown on examples from three application domains (Internet and cable television troubleshooting as well as call routing) using a reward function that takes both automation rate and handling time into account. We found that the offline techniques Escalator and Engager are able to improve performance mainly by reducing average handling time whereas EverywhereContender is also able to impact automation rate. All three techniques are still in the early stage of commercial implementation. Specifically, the *dynamic* versions of Escalator and EverywhereContender require extensive changes to conventional architecture of spoken dialog systems and have therefore not been used in any commercial application yet.

6. Expedients

[1] 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.

[2] E. Levin and R. Pieraccini, "Value-Based Optimal Decision for Dialog Systems," in *Proc. of the SLT*, Palm Beach, Aruba, 2006.

[3] A. Schmitt, M. Scholz, W. Minker, J. Liscombe, and D. Suendermann, "Is it Possible to Predict Task Completion in Automated Troubleshooters?," in *submitted to the Interspeech*, Makuhari, Japan, 2010.

[4] D. Suendermann, J. Liscombe, and R. Pieraccini, "Optimize the Obvious: Automatic Call Flow Generation," in *Proc. of the ICASSP*, Dallas, USA, 2010.

[5] D. Zimmerman, "A Note on Interpretation of the Paired-Samples t Test," *Journal of Educational and Behavioral Statistics*, vol. 22, no. 3, 1997.

[6] R. Sprinthall, *Basic Statistical Analysis*, Pearson Education Group, Upper Saddle River, USA, 2003.

[7] L. Kaelbling, M. Littman, and A. Moore, "Reinforcement Learning: A Survey," *Journal of Articial Intelligence Research*, vol. 4, 1996.

[8] S. Young, "Talking to Machines (Statistically Speaking)," in *Proc. of the ICSLP*, Denver, USA, 2002.