Dialog Methods for Improved Alphanumeric String Capture

Doug Peters¹, Peter Stubley¹

¹ Nuance Communications, Montreal, Canada
doug.peters@nuance.com, peter.stubley@nuance.com

Abstract
In this paper, we consider advances in automated over-the-phone alphanumeric string capture. For this task, acoustic confusions typically result in significant error rates. Of course, confusions also exist in human-to-human communication. However, humans employ dialog-level strategies with which to disambiguate confusions and correct errors – allowing high-fidelity transmission of alphanumeric strings across all but the noisiest of channels. These human strategies are examined and a subset amenable to automation is identified. The resulting automated error-correction dialog achieves 30% dialog error rate reduction compared to a conventional application in a high-volume commercial deployment. Further, the fact that there are many recognition errors in the context of a structurally simple dialog recommends this task for dialog optimization. We present an example of offline optimization and discuss the potential for online learning.

Index Terms: alphanumeric recognition, error correction, dialog optimization

1. Introduction
Worldwide deployments at Nuance Communications capture millions of alphanumeric strings per day. For this commercially important task, the accuracy of state-of-the-art automatic speech recognition (ASR) systems can still be unsatisfactory. Moreover, the underlying ASR technology is sufficiently mature that incremental ASR improvements are unlikely to provide significant gains.

Given an independence assumption, the string error rate, \( e_s \), is related to character error rate, \( e_c \), as \((1-e_s)=(1-e_c)^K\), where \( K \) is the length of the string. As a result, there are still important applications (with large \( K \)) for which a (string) error is more likely than none. Clearly, error-recovery is critical to high-fidelity alphanumeric string capture. Historically, the default error-recovery mechanism has been to simply request a repetition of the entire string. Automated capture is then abandoned if the caller declines the hypotheses offered for confirmation more than a few times. While somewhat effective, this strategy often fails short of desirable success rates, particularly if the string is longer than a few characters.

One clear improvement on this conventional strategy involves the integration of all available information from each repetition, a technique that has been developed in recent literature [1,2]. As effective as these methods are, they may still be insufficient to achieve desired levels of string capture in noisy conditions and large string length. In this paper, we additionally develop dialog strategies analogous to those used by humans in the transmission of alphanumeric strings over noisy acoustic channels. In particular, substring confirmation and correction appears critical to this success. That is, by partitioning the string into substrings or “chunks,” errors can be more easily identified and corrected.

Automated dialog optimization techniques have also been shown to provide robustness to recognition errors [3,4]. However, the application of these methods is not always practical [5]. When a Spoken Dialog System (SDS) is simple, and its component recognitions have low error rates, optimization may offer a relatively minor benefit. On the other hand, a SDS’s complexity can render optimization methods unwieldy. The proposed error-correction dialog for alphanumeric string capture represents a “middle ground,” perhaps ideally suited as a test-bed for dialog optimization methods, since the structure is relatively simple while the recognition task is very difficult. Consequently, we examine the optimization opportunities specific to this task. We also consider the extent to which recent automated methods of dialog optimization are appropriate.

In the following section, we examine the strategies that human interlocutors employ in the transmission of alphanumeric strings across a noisy channel, and formalize the components of an automated string-capture SDS to model these strategies. Then, converging on a candidate dialog structure built from these components, we describe in detail some of the practical considerations of the resulting dialog. Having reported on the effectiveness of these techniques in a large-scale public deployment in Section 4, we then examine the amenability of a string-capture error-correction dialog to existing methods of dialog optimization and report results of an example primitive optimization.

2. Dialog structure
The continuing challenge to alphanumeric string capture using ASR systems is the acoustic confusion inherent in the task, notably the “e-set” in English – i.e., \{b,c,d,e,g,p,t,v,z\}. But recognition errors in alphanumeric string transmission are by no means exclusive to ASR systems, being common to human-to-human oral communication, particularly over the phone. As a result, it is of interest to consider the strategies that human interlocutors employ to provide robustness to these errors, both in order to reduce transmission errors and to identify and recover from those errors.

Unfortunately, some of the methods that humans use for the present task are impractical for automation. Shared knowledge between speaker and listener, for example, permits a number of tricks that are beyond the current state-of-the-art. But even in the absence of such shared knowledge, human dialog strategies strongly depend on the flavor of alphanumeric string under consideration. In particular, length and vocabulary constraints are important. Table 1 gives four examples of commercial alphanumeric tasks with their candidate regular expressions.

<table>
<thead>
<tr>
<th>Application</th>
<th>Possible RegEx</th>
</tr>
</thead>
<tbody>
<tr>
<td>Password</td>
<td>[a-zA-Z0-9]*</td>
</tr>
<tr>
<td>Local Phone Number</td>
<td>[2-9][0-9]{1,6}</td>
</tr>
<tr>
<td>Reservation Number</td>
<td>[j-z][a-zA-Z0-9]{1,5}</td>
</tr>
<tr>
<td>Credit Card Number</td>
<td>[0-9]{1,16}</td>
</tr>
</tbody>
</table>

Table 1: constraints implicit in alphanumeric tasks.

For the purposes of discussion, we will consider those strings represented by the regular expression [a-zA-Z0-9]* – that is, we
will treat length constraints and letter case as extensions to what follows.
Relatively simple strings (such as local telephone numbers, using the less-confusable digits-only vocabulary) can be transmitted between humans without explicit confirmation. But reservation numbers, passwords, tracking numbers, and alphanumeric account numbers – especially those strings for which the cost of an error is high – often involve explicit confirmation (i.e., “let me read that back to you…”). An important exception to this rule occurs when an alphanumeric string is transmitted using “lexical guidance” or word-representations of the alphanumeric characters. Examples of this are tabulated below:

<table>
<thead>
<tr>
<th>Sub-strategy</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>radio alphabet</td>
<td>alpha bravo charlie</td>
</tr>
<tr>
<td>“as in” locations</td>
<td>a as in apple b like boy</td>
</tr>
<tr>
<td>common words</td>
<td>b i g (big)*</td>
</tr>
</tbody>
</table>

* when the string contains such words

Table 2: examples of lexical guidance.

If it were possible to force callers to provide alphanumeric strings according to the first two (i.e., the more common) variants of lexical guidance according to a pre-defined vocabulary, ASR would typically result in fewer errors. But even if we set aside the challenging task of maintaining a grammar to cover the vast majority of lexical examples that humans choose extemporaneously for “as in” locations, prompting for these locutions is only partly effective. If the prompt explicitly requires such locutions, our experience suggests that callers can too often abandon the call immediately. On the other hand, if these locations are strongly recommended it is rare to have much more than half of callers actually following this recommendation. Therefore, the case in which these strategies are not employed by the caller at the outset is still important. As a result, we consider the case in which there is no expectation of lexical guidance in the initial string transmission and where explicit confirmation does not feel unduly burdensome to callers.

In summary, we consider mostly-random (i.e., without the opportunity to invoke common words), confusable (i.e., tolerating explicit confirmation), unknown-length English alphanumeric strings, presented initially with no expectation of lexical guidance. While it may appear that this represents a small subset of alphanumeric strings of commercial interest, we claim that the resulting structure is sufficiently effective that it represents a foundation on which to build solutions for more general capture.

The upshot of these consideration constraints is simply that the dialog structure is now focused on confirmation and correction, for which we now consider human dialog strategies. Naturally, if there was a potential confusion in the original transmission of a string, there exists the potential for a similar confusion in its confirmation. Given that the cost of error is high, the lexical guidance strategy is common. As a result, we propose the use of “as in” locations in confirmation prompts for all but the least-confusable alphanumeric characters (i.e., “w” and “x”). While slowing the dialog down somewhat, this strategy provides a consistent user experience while enabling the level of accuracy that we require.

Another important dialog strategy employed by humans for the transmission of alphanumeric strings is the use of substrings (or “chunks”). While humans can use chunking in either transmission or confirmation, we will consider its application for confirmation exclusively. After all, in the same way that it is difficult to elicit initial string transmission with lexical guidance, it is also difficult to elicit initial string transmission with predictable substrings. An exception to this rule is the case of credit card numbers, for which the transmission is conventionally offered in chunks of four digits with pauses for confirmation.

An interesting thought experiment is useful to demonstrate the benefit of chunk-based confirmation. Consider the recognition lattice represented by the following table as if it were the recognition result for each utterance in a given dialog capturing a length-twelve alphanumeric account number:

<table>
<thead>
<tr>
<th>w</th>
<th>a</th>
<th>x</th>
<th>m</th>
<th>r</th>
<th>s</th>
<th>w</th>
<th>q</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>h</td>
<td>n</td>
<td>3</td>
<td>F</td>
<td>2</td>
<td>x</td>
<td>u</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 3: thought experiment recognition lattice.

Further, suppose that the score of the correct path through this lattice is random with respect to the other scores. That is, in this thought experiment, given a caller declines a recognition hypothesis offered for confirmation, the system requests a repetition, whose recognition result is precisely the same as that of the previous utterance(s) of the same (sub-)string. Using the conventional confirmation-of-the-entire-string, one would expect (using “expect” in its statistical sense) to present forty wrong strings to the caller before the correct one. With substring chunks of length three, however, we would expect the presentation of each chunk to involve a single wrong substring before the correct one for each of four chunks. The resulting dialog is therefore considerably snappier using chunked confirmation. Moreover, there is a real sense of progress when the caller hears and accepts each correct substring. Conventionally, there would be eighty dialog turns (forty declines and forty repetitions of the account number) before experiencing any progress. With substring confirmation, however, there is the expectation of progress every three turns, and success after only thirteen turns.

Clearly, chunk-length is an important consideration and should depend on the confidence of the individual characters in the received string. For long stretches of unconfusable characters, the chunk-length could be quite long. For confusable characters and in noisy environments, a shorter chunk is appropriate. A hierarchical chunking method (i.e., permitting confirmation with successively smaller chunks when long chunks are persistently declined) has also been implemented, but it is a relatively low-impact feature.
represent recognition states, and the arrows represent dialog actions conditioned according to the arc-labels. The labels Rej, and “no” represent (system) rejections and caller rejections (i.e., “declines”), respectively. The label “{corr}” represents a chunk-correction, and “no+” indicates a one-step correction.

In summary, we collect an unconstrained alphanumerical string which may be devoid of lexical guidance, and read it back to the caller in chunks with “as in” constructions for confirmation. The rhomboid in Figure 1 represents the mechanism to determine the “most sensible” next chunk, and its workings will be described in the following section. Once each chunk has been accepted (i.e., a “yes” is received from the caller) the caller is asked to confirm the subsequent chunk. When the material from the initial collection has been exhausted, the dialog ends.

Some interesting details are now presented in the context of a representative example dialog:

**Turn 1**: Please tell me your reservation code one letter or number at a time – r t w o c n i n e f h

Unfortunately, there is no word in English that is commonly understood to mean what mathematicians mean by the words “character” or “symbol.” The technical use of these words is largely unknown to the general public, and their use in such a dialog can be a source of some confusion.

**Turn 2**: Ok, let me read that back to you one part at a time; was the first <N> characters…? No

If the word “character” were generally understood in its technical sense, it might seem natural to prompt with “were the first <N> characters…?” With its common (mis)understanding, however, it may still be tempting to use “were the first <N>…?” But there are a number of reasons for not using this prompting (learned the hard way):

a) With lexical guidance, callers hear many more than <N> words, and this can be confusing.

b) It would be possible to create prompts like “…were the first two two two?” which result in caller distress.

c) Recognition errors can be insertions or deletions, in which case an explicitly fixed-length confirmation can cause confusion, involving an unwelcome cognitive load.

**Turn 3**: Please give me those first few again? – r t w o c

Since the accumulated hypothesis (see the following section for details) after the correction is the same as the original hypothesis (i.e., the ‘r’), there is no need to slow the dialog down with a lexical guidance (i.e., “as in Robert”) in the second confirmation attempt.

**Turn 4**: Was that ‘r, t w o, c as in Charlie?’ — yes

As mentioned earlier, it is difficult to cover all the “as in” examples that callers actually use. The distribution of their examples involves a long and creative tail. If the initial n-best list involved insertions or deletions, particularly an end-insertion, dialog-termination can be tricky to detect. By explicitly prompting for the “remainder,” the caller’s response helps us be confident in this decision.

**Turn 5**: was the remainder ‘nine, s as in Sam, h as in Harry’? – no: f as in Family

Rather than encourage such creativity by reading back the example used by the caller, we nudge them toward higher-running examples in the confirmation prompt.

### 3. Algorithmics

Fundamental to the error correction represented in the dialog example above is the use of all available evidence for hypotheses to be presented to the caller for confirmation. While sophisticated methods to combine this evidence have been described in the recent literature [1,2], a simple approximation provides near-optimal behavior in the context of alphanumerical strings. Briefly, we store the n-best list from the initial collection, and sum the log-likelihoods (“scores”) from the n-best lists of all correction utterances, discounting declined hypotheses (i.e., a “skip-list”). The only subtlety is how to handle hypotheses that do not appear in all the constituent n-best lists, illustrated artificially below:

<table>
<thead>
<tr>
<th>hyp1</th>
<th>score1</th>
<th>hyp2</th>
<th>score2</th>
</tr>
</thead>
<tbody>
<tr>
<td>a k m</td>
<td>-90</td>
<td>b s n</td>
<td>-90</td>
</tr>
<tr>
<td>a k n</td>
<td>-86</td>
<td>a k m</td>
<td>-97</td>
</tr>
<tr>
<td>b s n</td>
<td>-80 -Δ</td>
<td>a k m</td>
<td>-90 -Δ</td>
</tr>
</tbody>
</table>

Table 4: n-best score combination.

In effect, disjunctions between these two 3-best lists are added to those lists with a penalty Δ, which is set to be slightly larger than the active beam. Clearly, the optimal result is available with re-recognition of the two utterances using appropriately trimmed grammars, but with sufficiently rich n-best lists (common in alphanumerical string recognition), this simple combination results in the optimal hypothesis in the vast majority of situations.

The other important algorithmic consideration involves the chunking mechanism. Considerations governing chunk-length have been described above and we now describe the construction of a chunk’s “seed n-best list.” Given that the initially-collected n-best list can be represented as the map between string hypotheses and scores $M_{1} = \{h_{i}, s_{i}\}_{i=1...n}$, we derive the first chunk’s “seed n-best list” as follows. For nominal chunk-length $L$, $h_{i}(L)$ is assigned the score $s_{i}$, Then, for each subsequent hypothesis in $M_{2}$, its subset Levenshtein-aligned to $h_{i}(L)$ is considered. If this substring is not already represented in the chunk n-best list, it is added with the score of its parent string from $M_{2}$.

Once a chunk-hypothesis has been accepted by the caller, that result is then Levenshtein-aligned to all the original hypotheses in $M_{2}$, and the subsequent chunk is seeded by an n-best list derived in the same manner.

We also provide for “uncooperative” caller behavior. That is, if presented with an erroneous chunk-hypothesis, callers can (and occasionally do) correct that chunk with a substring of a different length. That is, they might correct only the error (as in Turn 5 of the example dialog above), or they might even correct beyond the length of the offered chunk. In the first case, we can interpret the correction if it is found in the active n-best list, re-prompting otherwise. In the second case, we can reconstruct the n-best list using the chunk-size imposed by the caller before combining the n-best list representing their correction.

### 4. Results from deployment

The dialog mechanisms described above have been deployed in a large-scale commercial SDS. Moreover, a conventional dialog was simultaneously deployed, with calls assigned randomly between them. The relative gains of the proposed dialog relative to the conventional SDS are summarized in Table 5 below. Naturally, the performance of the two SDSs is identical in the no-correction-required category. With respect to error correction, on the other hand, the relative success of the new dialog is quite significant, and more than makes up for the increase in dialog failures as can be seen from the overall gains of 30% relative.
### Table 5: relative dialog performance in deployment.

<table>
<thead>
<tr>
<th>Category</th>
<th>Gain</th>
</tr>
</thead>
<tbody>
<tr>
<td>Error correction</td>
<td>+50%</td>
</tr>
<tr>
<td>Dialog failure</td>
<td>-56%</td>
</tr>
<tr>
<td>Agent request</td>
<td>+55%</td>
</tr>
<tr>
<td>Overall error</td>
<td>+30%</td>
</tr>
</tbody>
</table>

The dialog failure losses are due to three factors:

1. The proposed dialog is considerably more complex than its predecessor.
2. The joint-optimization method adopted is susceptible to divergence when the caller is “uncooperative” – e.g., when a caller declines a correctly-recognized chunk.
3. The required max-decline/ max-no-speech/ max-help thresholds may have been too tight – having been chosen as appropriate to more conventional SDSs.

Finally, the significant reduction in agent requests suggests that the proposed dialog is less frustrating for callers than the legacy SDS.

## 5. Dialog optimization

Recent developments in dialog optimization attempt to find the optimal action (i.e., decision) policy given a fixed dialog structure and a set of features representing the current dialog state. Often in the flavor of reinforcement learning [3], these methods optimize a pre-specified objective function. It is sometimes non-trivial, however, to formalize dialog requirements into an appropriate objective function. For example, it is clear that the error-out decisions above (i.e., the max-* thresholds resulting in dialog failures) are candidates for such techniques. However, the reason for these thresholds, namely, the reduction of caller frustration, is difficult to measure. While it is clear that caller frustration mounts as dialogs lengthen, it is not clear that aborting a dialog in the final turns of correcting the final chunk of a long string is a real solution to that frustration.

The yes- and no-labeled arcs in the Figure 1 represent decisions offering little opportunity for optimization, as these decisions are near-optimal with no rejection at all. Elsewhere in the dialog, however, rejection decisions are more obvious candidates for optimization, and in particular that decision at the initial collection state. However, observations available from the analysis of logs from the deployed SDSs provide an interesting opportunity for an explicit offline optimization, as follows.

The probability of success for the dialog depends on the initial threshold, \( \theta \), as:

\[
p_d(S) = [p_\theta(A) + p_\theta(R)p_\theta(A|R)] p_\theta(S|A)
\]

where the symbols A, R, and S represent accept, reject and success, respectively. Since the new dialog was initially deployed with a very low rejection threshold, log analysis provides good estimates of \( p_\theta(R), p_\theta(A) = 1 - p_\theta(R), \) and \( p_\theta(S|A) \) for all practical values of \( \theta \). On the other hand, the conventional deployment offers measurements of \( p_\theta(A|R) \) – that is, the probability that a subsequent dialog turn will yield an acceptable utterance when the previous utterance was rejected – for low values of \( \theta \). For higher values of \( \theta \), however, we claim that a decline-repeat – i.e., a repeated utterance after the caller rejection of a confirmation string – is sufficiently similar to a reject-repeat – i.e., the repeated utterance after the system rejection – to permit the estimate of \( p_\theta(A|R) \) over the \( \theta \)-range of interest.

The resulting relative dialog success gains (i.e., a candidate objective function for this decision threshold) are illustrated vs. initial rejection threshold in Figure 2 below:

![Figure 2: threshold dependence of dialog success](image)

What we observe, however, is that the optimal settings for the confidence thresholds here (and in other states) are very close to the settings intuitively selected by voice user interface designers (15 in the present case). Consequently, the value of many of these optimizations is modest.

## 6. Conclusions

In this paper, we present dialog methods for the improved capture of alphanumeric strings using ASR. Substring-based confirmation and correction with lexical guidance emulates the most common human strategy for achieving such capture over a noisy acoustic channel. This dialog structure, coupled with the optimal extraction of recognition hypotheses from the integration of all available evidence has been shown to be very effective, yielding 30% relative improvement in the dialog error rate.

We also examine the opportunities for dialog optimization. While online learning was not present in the initial deployment, we show an example of practical offline dialog optimization. Due to the commercial importance of this task, the relative simplicity of the dialog structure and the high recognition error rates, we had hypothesized that alphanumeric string capture would represent an ideal test-bed for dialog optimization research. However, we found that the available optimizations yield modest gains relative to an initial design due to industry experts.

## 7. References