RI 04003 11 May 2004 Computer Science

IBM Research Report

SAMVAAD: Speech Application Made Viable For Access-Anywhere Devices

Pankaj Kankar
IBM Research Division
IBM India Research Lab
Block I, I.I.T. Campus, Hauz Khas
New Delhi - 110016. India.

Mohit Kumar
IBM Research Division
IBM India Research Lab
Block I, I.I.T. Campus, Hauz Khas
New Delhi - 110016. India.

Amit Anil Nanavati
IBM Research Division
IBM India Research Lab
Block I, I.I.T. Campus, Hauz Khas
New Delhi - 110016. India.

Nitendra Rajput
IBM Research Division
IBM India Research Lab
Block I, I.I.T. Campus, Hauz Khas
New Delhi - 110016. India.

ABSTRACT

The proliferation of pervasive devices call for the enablement of ubiquitous access via multiple modalities. Since the capabilities of pervasive devices differ vastly, device-specific application adaptation becomes a necessity. We address the problem of speech application adaptation by dialog call-flow reorganisation for pervasive devices with different memory constraints. One usability criterion we use is minimising the number of dialogs in the call-flow. Given an *atomic* dialog call-flow C and memory size m, we present optimal algorithms, RESEQUENCE and BALANCETREE, that output the corresponding reorganised version C_m, such that the number of questions (prompts) is minimum. In some cases, an 'ideal' reference call-flow that takes into account various usability criteria may be available. In such cases, minimising the number of changes to this ideal call-flow to accommodate memory-constrained devices forms another 'usability criterion'. We present two algorithms, MASQ and MATREE that minimise the distance from the ideal call-flow.

The following observation forms the cornerstone of all the algorithms in this paper: Two grammars g_1 and g_2 comprising of $|g_1|$ and $|g_2|$ elements respectively can be merged into a single grammar $g = g_1 \times g_2$ having $|g_1| \cdot |g_2|$ elements for the sequential case, and $g = g_1 + g_2$ having $|g_1| + |g_2|$ elements for the tree case.

We introduce the concept of an $\langle m, q\rangle$*-characterisation* of a call-flow, defined as the set of pairs $\{\, (m_i, q_i) \mid i \in N\}$, where q_i is the minimum number of questions required for memory size m_i. Each call-flow has a unique, *device-independent* signature in its $\langle m, q\rangle$-*characterisation*, which provides a means for comparing call-flows from an adaptation standpoint.

Keywords

Speech Processing, Pervasive Computing, Algorithms, Dialog Call-Flows, Call-Flow Optimisation, Conversational Interface Usability

1. INTRODUCTION

Consider a user accessing the *Theatre-near-you* interactive voice response (IVR) system to buy movie tickets from his car phone while driving to the theatre. As a result of his mobility, he might get disconnected from the IVR system at any time during the conversation. Ideally, he would prefer that the particular call-flow is available on his device, and access to a remote server happens only when required.

The proliferation of pervasive devices call for the enablement of ubiquitous access via multiple modalities. Users are increasingly accessing remote applications on the internet and running a plethora of local applications from their mobile devices. From the users' point of view, they would like more and more applications to be accessible via various interfaces (voice, multimodal) from their pervasive devices.

Pervasive devices are different from desktop computers in two fundamental ways. One, they occur in various sizes with vastly differing capabilities, and by virtue of mobility, are not always connected to the network. This combination gives rise to some very interesting challenges and possibilities.

From the application provider's point of view, he has to provide alternatives for various devices since there is no one-size-fits-all solution. This means that he has to deploy and maintain various incarnations of software that essentially provides the same functionality. An application author encounters the M times N problem, namely, an application composed on M pages to be accessed via N devices requires M x N authoring steps, and results in M x N presentation pages to be maintained.

To address the application developer's nightmare, many application programming tools have been proposed and are available [1]. Such tools allow the programmer to develop a generic application which is automatically adapted by the tool for various devices profiles. However these techniques address problems of applications in the visual domain.

Speech interfaces provide a natural form of interaction and enable access through devices with limited text/visual capabilities. A significant improvement in speech recognition [7, 8] and speech synthesis [2] technologies over the last decade makes speech interface to applications practicable. Conversational systems leverage domain knowledge to improve on the recognition accuracy. In a conversational system, a dialog call-flow defines a sequence of questions and possible answers for the interaction.

We are interested in adapting voice applications automatically for various devices. Traditionally, voice applications run on a remote server, and several client-server interactions take place in the course of a dialog. A client-server model incurs transmission costs and is prone to transmission errors. This may result in degraded speech recognition accuracy. The use of compression for reducing transmission costs introduces other complications [9]. In order to circumvent such problems, speech recognition at the client offers a viable alternative. For supporting client-side speech recognition on a variety of devices with *limited resources*, device-specific application adaptation is essential. However, where connectivity is an issue, it may be more convenient to download the entire application to the device and process speech locally. This requires designing a device-specific voice application. A similar demand comes from embedded applications where devices with different capabilities process inputs locally. We address the problem of device-sensitive dialog call-flow[1] adaptation. The basic idea is to manipulate grammars of dialog call-flows in order to satisfy memory constraints.

Devices vary considerably in terms of their memory capacities: A smartcard typically has memory of the order of a few tens of KBs. Cell phones have few MBs of memory and PDAs have memories in tens of MBs. There is a lot of variation in the memory requirements of dialogs too. Within a call-flow, an address identification dialog could require a vocabulary of a few thousand words, whereas a credit card number dialog needs a vocabulary size of about ten words. In some cases, it may be necessary to break the dialog into a few small subdialogs to accommodate a device limitation, and in others, a few small subdialogs may be combined as long as the dialog can be handled efficiently.

1.1 Related Work

Litman and Pan [4] discuss about improving performance by adapting dialog behavior to individual users. They claim that the performance of the system can be improved if the user can effectively adapt the system's behavior, so that the system will use the dialog strategies that best match the user's needs at any point in a dialog. Jameson [3] discusses the cognitive aspects of conversational applications taking the available user time into consideration and the extent to which the user can concentrate on the interaction. Dialog call-flow adaptation of a conversational system for improving the speech recognition accuracy has been addressed in [5]. This work does not take the device characteristics into consideration.

Levin et al. [6] formalise the problem of dialog design as an optimisation problem with an objective function reflecting different dialog dimensions relevant for a given application. They show that any dialog system can be formally described as a sequential decision process in terms of its state space, action set, and strategy.

[1]or simply call-flow

With additional assumptions about the state transition probabilities and cost assignment, a dialog system can be mapped to a Markov decision process (MDP). They present a variety of data driven algorithms for finding the optimal strategy based on reinforcement learning.

1.2 Our Contribution

We investigate the problem of dialog call-flow reorganisation for pervasive devices with memory size restrictions. The crux of the reorganisation lies in altering the memory requirements of the underlying grammar. We achieve this by merging *atomic*[2] grammars while minimising the number of questions, thus accounting for one aspect of usability. Usage of the number of questions as a usability criterion is consistent with the efforts for evaluating spoken dialog systems [12]. We present deterministic algorithms, RESEQUENCE and BALANCETREE, which provide optimal solutions for the two types of call-flows, *sequential* and *tree-type* respectively.

In some cases, an 'ideal' reference call-flow that takes into account various usability criteria may be available. In such cases, minimising the number of changes to this ideal call-flow to accommodate memory-constrained devices forms another 'usability criterion'. We present algorithms, MASQ and MATREE that minimally alter a given dialog call-flow to accommodate it within a given memory constraint.

A derivative of our approach is a *device-independent* characterisation of dialog call-flows. This signature can be constructed by finding the set of ⟨memory, minimum number of questions⟩ corresponding to each call-flow.

A direct implication of our work is the realisation of automated adaptation tools for building speech applications for memory-constrained pervasive devices.

Section 2 presents the assumptions and the setting in which the problem is addressed. Section 3 presents the details of RESEQUENCE, BALANCETREE, MASQ and MATREE including an ⟨m,q⟩-*characterisation* of an example call-flow. Section 4 connects theory to practice by examining actual device capabilities and current recognition technology in the domain. Section 5 concludes the paper by discussing future work and challenges.

2. PROBLEM DESCRIPTION

In this section we provide a brief description of a speech recognition system and the memory required in different portions of this system. We also describe the issues that are involved in design of a dialog call-flow.

2.1 Memory Requirements of Speech Recognition Systems

An Automatic Speech Recognition (ASR) system consists of two main components, an acoustic model and a language model. The acoustic model of an ASR system models how a given word or "phone" is pronounced. The acoustic model of a statistical speech recognition system generates the likelihood of the input speech signal with a hypothesised sentence. Hidden Markov Model and Neural Network are the most common techniques for acoustic modelling of ASR systems.The language model provides a probabilistic estimate of the likelihood of the sentence hypothesis. The memory requirement of an acoustic model is dependent on the size of the acoustic model and the code storage. Similarly, the memory required by a language model is dependent on the code storage and on the vocabulary size of the model on which it has been trained. The most common technique used for this purpose is an N-gram language model. An N-gram model provides the probability of N^{th} word in a sequence, given a history of $N-1$ words. In conversational systems, a language model is replaced by a speech recognition grammar. This grammar represents the possible phrase/sentence that are expected as possible speech input. In such cases, the perplexity[3] and size of grammar determine the memory required by a language model. In subsequent sections, we will discuss the memory required by an ASR with varying grammar sizes. For the discussions to follow, we assume in this paper that the acoustic model is the same for all grammar sizes. Hence the variation in memory requirement of an ASR is due the varying grammar size.

[2]An atomic grammar is one which cannot be split into subgrammars.

2.2 Dialog Call-flows

Here we present various practical considerations involved in designing a call-flow, and our corresponding simplifying assumptions. A call-flow consists of a question-answer sequence. Questions are presented to a user by playing the corresponding audio on a device. Answers captured from the user are processed by a speech recognition system. We focus on the speech recognition requirements of a call-flow.

Call-Flow Types: Call-flow applications can be divided into two types, *sequential* and *tree-type*. In a sequential call-flow, the next question asked is independent of the answer to the current question. An example is a purchase application that asks for the receiver's address and the sender's credit card information. Each node in this type of call-flow acts as a input block. A tree-type call-flow is one where a user response determines the next question. Each question in this call-flow acts as a decision block, which results in a tree topology.

We present mechanisms that alter the memory requirement of a call-flow by changing the underlying grammars. In practice, such reorganisation of dialogs in a call-flow needs to take into account several factors, some of which are presented below along with our assumptions and arguments:

1. *Natural language grammars:* There can be cases wherein a grammar has fewer choices, but its representation in natural language sentences increases the perplexity. In such a case, the memory required to process the grammar would not be dependent on the number of choices, but on its perplexity. For example, there can be two grammars, g_1 and g_2 that have 4 choices, {*red, blue, green, white*} and {*one, two, three, four*} respectively. However, the perplexity of g_1 can be higher as this grammar allows choices such as {I want <color>, Give me <color>, Let me have <color>, Please give me <color>} as its possible natural language sentences. Here <color> can be substituted by any of the {*red, blue, green, white*}. However g_2 is a 4-digit grammar which has no natural language constructs. Therefore the perplexity of g_1 will be much higher than that of g_2, even though both have equal number of choices. Such natural language representations are heavily dependent on the design of a grammar and are bound to have implementational dependencies. Assumption 1: The perplexity is independent of the natural language representation of the grammar.

2. *Multiple grammars:* There could be applications which require the presence of more than one grammar at a particular time. For example, there are applications that use a global grammar to handle some of the common user responses like

[3]The perplexity $|g|$ of a grammar g is equal to the number of constructs in the language that satisfy the grammar.

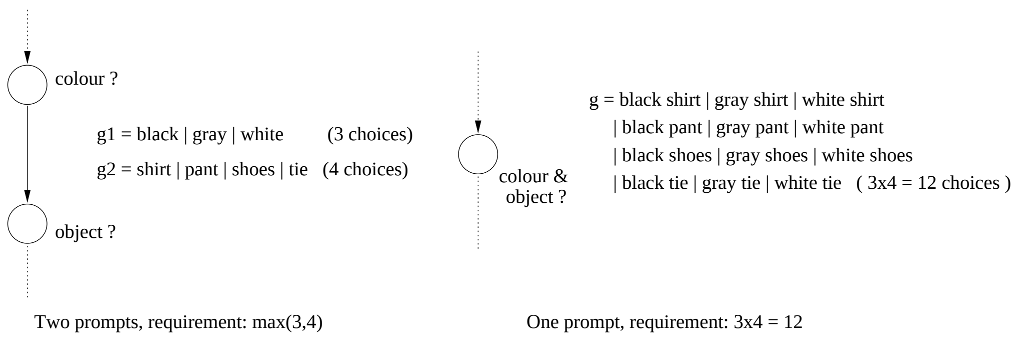
colour ?
g1 = black | gray | white (3 choices)
g2 = shirt | pant | shoes | tie (4 choices)
object ?
Two prompts, requirement: max(3,4)
g = black shirt | gray shirt | white shirt
| black pant | gray pant | white pant
| black shoes | gray shoes | white shoes
| black tie | gray tie | white tie (3x4 = 12 choices)
colour & object ?
One prompt, requirement: 3x4 = 12

Figure 1: Effect of merging/splitting a sequential grammar.

"go back", "repeat" and "exit". Call-flow optimisation in presence of multiple grammars would require a detailed understanding of each call-flow separately.
Assumption 2: The call-flow has only a single grammar active at a particular time.

3. *Usability Constraints:* Usability constraints can be of several types. They could be time-based "the total running time of the application should not exceed T", comfort-based "no more than p questions should be asked", "this particular question should *not* be split or merged with another question". Note from the above examples that these constraints might apply globally (entire call-flow) or locally (single dialog). We restrict ourselves to considering the maximum number of questions as a constraint.
Assumption 3: All constraints can be interpreted in terms of allowing/enforcing merge/split operations on a subsequence of questions.

4. *Best Practices:* As a result of optimisation for different devices, the same application may have a different call-flow sequence when accessed on different devices. This may not be considered as a user friendly interaction model, even though it uses the device resources optimally.
Argument 4: The fact that alteration makes it *possible* to run an application on a device is of paramount importance.

Finding an optimal solution while considering the above speech-specific nuances is a challenging task. Our assumptions admit a precise formulation for gaining further insights into the problem. Although, the algorithms in this paper do not take into account the above factors explicitly, we claim that all assumptions (except 2 above) can be translated into reorganisation constraints that can be input to our algorithms.

In the next section, we present the reorganisation constraints used by the algorithms presented in section 3.

2.3 Reorganisation Constraints

For reducing the number of questions to improve usability and make optimum use of memory resources we merge the dialogs and their corresponding grammars. However in most of the real life applications, grammar merges across certain subdialogs may not be possible due to various constraints. These constraints can be due to usability, data dependency, server round trip or user imposed. We call these constraints as reorganisational constraints. While merging dialogs, the reorganisational algorithms will have to work under the provided reorganization constraints.

Reorganisational constraints represent *hard* constraints of two types. One, that insist that a certain group of dialogs be merged, and two, that forbid a certain group of dialogs from being merged. The first set *must-merge* is a set of sets of dialogs which must be merged. The second set *must-separate* is a set of sets of dialogs which must not be merged. The former constraints are handled by a preprocessing step where all those dialogs that "have to" be merged, are merged. The new dialog call-flow generated by this merging of dialogs is then considered as an atomic call-flow. The reorganisation algorithms mentioned in section 3 are applied over this call-flow. *T*ype 2 constraints are incorporated in the algorithm, as will be explained in the next section. The call-flow of an example flight reservation application is shown in 9. We will itemize the various reorganisational constraints and their significance with respect to this call-flow.

- **Data dependency/validation Constraints** *Departure city can not be merged with arrival city.* The grammar for arrival city depends on the departure city. On getting the user response for departure city, the application goes back to the server and fetches the list of valid arrival cities. Since the call flow needs to fetch data from the server based on user response for departure city, the departure city subdialog can not be merged with the next subdialog.

- **Usability constraints** *Not more than four atomic subdialogs can be merged* To maintain the usability of the dialog system, the call-flow should not ask for more than four pieces of information from the user in one go.

- **User constraints** *Date should not merged with airline*
Flight number should not be merged with credit card type
If merged, day and month should always occur together If merged, expiry month and year should occur together.

There may be constraints due to several other reasons in a real-life application but they can be modeled as reorganisational constraints. Thus the system can take care of practical limitations on merging of dialogs while doing the reorganization. These constraints help in specifying the usability requirements in the process of reorganising the call-flow.

2.3.1 Implications

The effect of reorganization constraints is that they forbid certain subdialogs to merge even if they satisfy the memory requirement. These constraints provide a mechanism to represent the usability factors in the algorithms presented in section 3. We also show an example of how the reorganisation graph of a call-flow changes due to these constraints.

3. REORGANISATION ALGORITHMS

We assume that the input call-flow comprises of *atomic* dialogs[4] only. Such a call-flow is called an *atomic call-flow*. An atomic call-flow has the least memory requirements and also the most number of dialogs.

We present two sets of algorithms. The first set consists of two algorithms RESEQUENCE and BALANCETREE that minimise the number of dialogs (questions/prompts) given an atomic dialog call-flow with respect to a given memory constraint and operate on sequential and tree-type call-flows respectively. We use L for a sequential call-flow, T for tree-type, and C includes both types. M denotes the memory constraint.

Apart from the memory constraint, all algorithms accommodate *reorganisation* constraints. Reorganisation constraints represent *hard* constraints that cannot be violated (Please refer to section 2.3 for details.)

The second set of algorithms MASQ and MATREE *minimally alter* the given reference call-flow (sequential and tree-type respectively) to accommodate the given memory constraint. A reference call-flow C^r may be the result of a myriad of considerations and serves as a guideline for reorganisation. C^r represents a guideline and therefore is a *soft* constraint. It need not be atomic. For this set of algorithms, we naturally require a notion of distance to quantify minimal alteration.

3.1 Minimising the Number of Dialogs

In this section, we present RESEQUENCE and BALANCETREE to minimise the number of dialogs in a call-flow while respecting the memory and reorganisational constraints.

3.1.1 Sequential Call-flows

The following observation forms the operating handle for RESEQUENCE in handling sequential call-flows.

Observation 1. Two grammars g_1 and g_2 comprising of $|g_1|$ and $|g_2|$ elements respectively can be merged into a single grammar $g = g_1 \times g_2$ having $|g_1| \cdot |g_2|$ elements. Figure 1 shows an example, where as a result of a merge operation the memory requirement goes upto 12 from 4.

A call-flow can be represented by a sequence $L = \{1, \ldots, n\}$ of *atomic* dialogs representing the order in which the dialogs are presented. The goal is to merge as many questions as possible while respecting the memory constraint. The memory requirement $m(g_i)$ for each g_i is known. We construct a graph G as follows. The vertex set $V(G)$ contains precisely the elements of L. For each vertex i in G, we add edge (i, j) if $\Pi_{k=i}^{j} m(g_k) \leq M$ $(i < j \leq n)$, i.e, the memory requirement of the merged grammars g_i through g_j can be accommodated within memory constraint M. As a result of this, G becomes a directed acyclic graph. Note that G could be disconnected. Now, we need to find the shortest path (or set of paths) from 1 to n, by finding the shortest path for each connected component of G. Each edge in the shortest path (set of paths) denote the subsequence of questions being merged. Figure 2 details RESEQUENCE. The sets of dialogs in *must-merge* are merged as a preprocessing step, and dialogs merged as a result of this step are *considered atomic*. L_m denotes the output call-flow with the minimum number of dialogs. L_m may contain merged (non-atomic) dialogs.

Figure 3 shows an example of a graph with 7 nodes. The edges of this graph represent the possible merges in the call-flow. The dotted edges identify the nodes that are not allowed to merge due to reorganisation constraints. The shortest path for the graph is indicated by thick edges in Figure 3.

[4]dialogs which cannot be split into subdialogs - analogous to atomic grammars

```
1. input: atomic sequential call-flow $L_a$.
2. output: sequential call-flow $L_m$ with the minimum number of questions.
3. Construct a graph $G(V, E)$ as follows:
   (a) Merge all must-merge dialogs in $L_a$ to obtain $L_a^m$.
   (b) Represent all dialogs by vertices labelled {1,...,n}
   (c) for each vertex $i (1 \leq i \leq n)$
       i. for each vertex $j (i \leq j \leq n)$
       ii. if $(\Pi_{x=i}^{j} m(g_x) \leq M)$ && $\{i, j\} \notin$ must-separate,
           add $(i, j)$ to $G$.
4. Find the shortest (set of) path(s) as follows:
   (a) start = 1. $L_m = \emptyset$.
   (b) while $(start \leq n)$
       i. $L_m = L_m \cup \{start\}$.
       ii. select $max_j$ such that $(start, j) \in E$.
       iii. start = j+1.
5. output $L_m$.
```

Figure 2: RESEQUENCE – An optimal shortest path algorithm.

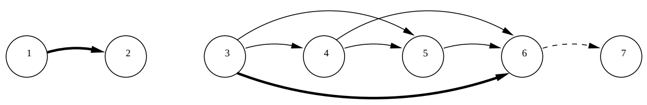

Figure 3: An example directed acyclic graph of a call-flow.

CLAIM 1. RESEQUENCE *is correct and runs in $O(n^2)$ time. The graph construction phase takes $O(n^2)$ time to check every pair of vertices for adding edges. The shortest path phase takes $O(n)$ time, since at each vertex the largest adjacent vertex can be chosen greedily to yield the shortest path.*

3.1.2 Tree-type Call-flows

The following observation forms the operating handle for BALANCETREE to process tree-type call-flows.

Observation 2. Two grammars g_1 and g_2 comprising of $|g_1|$ and $|g_2|$ elements respectively can be merged into a single grammar $g = g_1 + g_2$ having $|g_1| + |g_2|$ elements. Figure 4 shows an example. As a result of the merge operation, the memory requirement goes up from 2 to 4 ($g1$ to $g1'$).

Definition 1. The *degree* of a vertex is the number of its children.

Definition 2. A *2-subtree* of a vertex v is a tree of depth 2 with v as the root.

Definition 3. A 2-subtree of a vertex v is *balanced* if all the leaves of the 2-subtree are at distance 2 from v, i.e., no child of v is childless. A 2-subtree of a vertex v is *1-balanced* if at least one child of v is childless. A 2-subtree is either balanced or 1-balanced.

Definition 4. Let the maximum degree of any vertex in a call-flow tree be denoted by Δ. The *vacancy* of a vertex v is defined as (Δ - *degree(v)*).

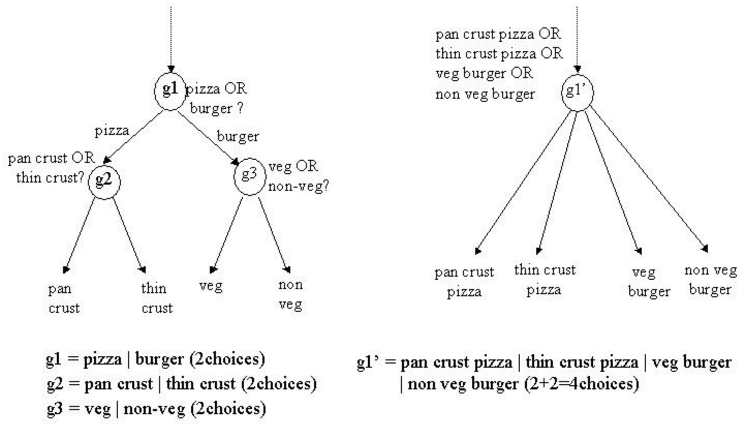
pizza OR burger ?
pizza
burger
pan crust OR thin crust?
veg OR non-veg?
g1
g2
g3
pan crust
thin crust
veg
non veg
g1 = pizza | burger (2choices)
g2 = pan crust | thin crust (2choices)
g3 = veg | non-veg (2choices)
pan crust pizza OR thin crust pizza OR veg burger OR non veg burger
g1'
pan crust pizza
thin crust pizza
veg burger
non veg burger
g1' = pan crust pizza | thin crust pizza | veg burger | non veg burger (2+2=4choices)

Figure 4: Effect of merging/splitting a tree-type grammar.

```
1. input: tree-type call-flow $T$.
2. output: tree-type call-flow $T_m$ with the minimum number of questions.
3. initialise $T_m = T$; boolean changed = false.
4. do
    (a) Find the longest path in $T_m$ and identify its lowest 2-subtree $t_2$.
    (b) if (shorten($T_m$, $t_2$)) changed = true.
5. while (changed)
6. output $T_m$.

7. shorten($T_m$, $t_2$)
    (a) while ($t_2 \neq$ "root")
    (b) do
        i. if (fold($t_2$)) return true.
        ii. else $t_2 = parent(t_2)$.
    (c) done
    (d) return false.

8. fold($t_2$)
    (a) if ((vacancy($t_2$) $\geq$ degree(children($t_2$))) return true.
    (b) return false.
```

Figure 5: BALANCETREE: An optimal shortest tree algorithm.

Definition 5. The *fold* operation is defined on the root v of a 2-subtree and allows v to directly inherit all its grandchildren if the *vacancy*$(v) \geq \Sigma_i$*degree(child*$_i(v)$*)*. As a result of this operation, all the grandchildren of v become its own children, and the original children are removed. This operation reduces the height of the tree by 1.

CLAIM 2. *The greedy application of the folding operation cannot lead to suboptimal solutions.*

PROOF OUTLINE. A greedy application of the folding operation on the root of a 2-subtree v can lead to two possibilities. One, inspite of this *fold(v)* operation, in a subsequent step *fold(parent(v))* is still possible, or two, that it is not. In the first case, since both *fold* operations must be done optimality is preserved. In the second case, it turns out that only one of *fold(v)* or *fold(parent(v))* could have been applied, either of which would lead to a height reduction of 1. $\square$

Claim 2 suggests that a bottom-up approach on the longest paths in the tree one 2-subtree at a time might provide a solution. This is the essence of BALANCETREE(figure 5). At each step, the longest path is found, its height reduced by 1, if a *fold* operation is possible at any vertex from the grandparent of the leaf in the longest path to the root. Notice that *shorten* traverses up the tree till it is able to reduce the height by 1. After this reduction, the longest path is calculated again and the same procedure is applied. If at any time, the longest path cannot be reduced, the algorithm terminates. Since the longest path is found globally at each step, and since the height of the tree is reduced only 1 at a time, we obtain a maximal height reduction. It follows that:

CLAIM 3. BALANCETREE *is correct and runs in $O(n^2)$ time where n is the number of vertices in the tree. Since the fold operation is the dominating cost, consider the degenerate case of a tree of depth n with one vertex at each level (a path). Suppose the root has vacancy n, then each vertex folds into its parent bottom-up one at a time. This accounts for $O(n^2)$ fold operations. Each vertex is examined a maximum of 2 times for each level it visits, as a child for its degree and as a parent for its vacancy amounting to a cost of $2n^2$.*

3.1.3 *Hybrid Call-flows*

In general, *hybrid* call-flows may contain sequential parts as well as tree-type parts. The algorithms RESEQUENCE and BALANCETREE can operate on the separate parts independently of each other. Without loss of generality, we can execute RESEQUENCE followed by BALANCETREE. Note that as a result of RESEQUENCE, a shortened sequence may contain a vertex v with increased memory requirements and hence a reduction in *vacancy(parent(v))*. This reduction in vacancy may prevent v from folding into its parent. If RESEQUENCE would not have affected v, then BALANCETREE would have folded v into its parent. Either case leads to a height reduction of 1. This argument is similar to the one used in claim 2 above.

3.2 Minimally Altered Call-flows

Minimising the number of questions need not be the single motivating factor for reorganisation. The design of a call-flow (without memory restrictions) typically takes into account numerous factors including speech recognition accuracy and natural language processing among others. Using such a call-flow as a reference and adhering to specified reorganisational constraints, it is possible to design algorithms that *minimally* alter the reference call-flow to meet memory restrictions.

This section presents the minimally altered counterparts MASQ and MATREE of RESEQUENCE and BALANCETREE respectively.

For quantifying minimality, it is necessary to define a notion of distance. We introduce a simple notion of distance based on two operations: *merge* and *split*. A single application of either of these operations on a call-flow C (whether sequential or tree-type) increases the distance of the modified version from C by 1. Let C_a denote the atomic version of C. A *split* operation on a non-atomic call-flow C can be simulated by replacing a dialog in C by its atomic components from C_a. Observe that no *split* operation was required for minimising the number of dialogs in a call-flow because the initial call-flow was atomic. In this case, however, since the reference call-flow need not be atomic, we need to support the *split* operation.

3.2.1 MASQ

We have the following question: Given a sequential, not necessarily atomic reference call-flow L^r and a memory constraint M, find a call-flow L^r_m, a *minimally altered* version of L^r that satisfies the memory constraint M.

MASQ is simple. If any dialog can be accommodated within M, it remains unchanged. For the others, it has to be split. Figure 6 details the algorithm.

CLAIM 4. *MASQ is correct and efficient.*

PROOF OUTLINE. The basic idea is to split only those dialogs that use memory larger than M. Further, the smallest number of splits are ensured by the *split* routine. Each call to *split* involves a linear search of the corresponding atomic component set. This greedy method yields an optimal solution. □

3.2.2 MATREE

MATREE works on tree-type call-flows. In this case, the 'splitting' of a vertex is like an 'unfolding' (similar to the *fold* operation being analogous to the *merge*). Figure 7 describes the algorithm. Note that the *unfolding* operation may cause the depth of the tree to increase, but the memory requirement at the vertex decreases.

CLAIM 5. MATREE *produces a tree with the minimum number of alterations.*

```
1. input: atomic sequential call-flow $L_a$.
2. input: reference sequential call-flow $L^r$.
3. output: minimally altered sequential call-flow $L^r_m$.
4. Construct a graph $G(V, E)$ as follows:
    (a) Represent all dialogs by vertices labelled {1,...,n}
    (b) Let $L^r_m = L^r$.
    (c) for each vertex $i (1 \le i \le n)$
    (d) if $(m(g_i) > M)$, then $split(i, L^r_m)$.
5. output $L^r_m$.

6. split$(v, L^r_m)$
    (a) Find the set of atomic components $S_v = \{v_1, ..., v_\ell\}$ of $v$
        from $L_a$.
    (b) If $\exists v_i \in S_v, m(v_i) > M$, output "IMPOSSIBLE" and exit.
    (c) Otherwise,
          i. $i = 1. SS_v = \emptyset$.
         ii. Find the largest $k$ such that $\Pi_i^k m(v_j) \le M$.
        iii. $i = (k + 1)$.
             $SS_v = SS_v \cup \{i - k\}$.
             if $(k < \ell)$, repeat previous step.
    (d) Replace $v$ by $SS_v$.
```

Figure 6: MASQ: Minimally Altered Sequential Call-Flow Algorithm.

```
matree$(T_a, T^r)$
1. input: atomic tree-type call-flow $T_a$.
2. input: reference tree-type call-flow $T^r$.
3. output: minimally altered tree-type call-flow $T^r_m$.
4. initialise $T^r_m = T^r$;
5. do
    (a) Traverse $T^r_m$ in preorder and if $(m(T^r_m) > M)$, then
          i. unfold$(T^r_m)$ ;
         ii. matree$(T_a, T^r_m)$ ;
6. output $T^r_m$.

7. unfold$(T)$
    (a) Identify the corresponding subtree $T'$ of $T$ in $T_a$.
    (b) Traverse $T'$ bottom-up and for each vertex $v' \in T'$,
        $T'_m$ = shorten$(T', v)$.
    (c) Replace $T$ with $T'_m$.
```

Figure 7: MATREE: Minimally Altered Tree Algorithm.

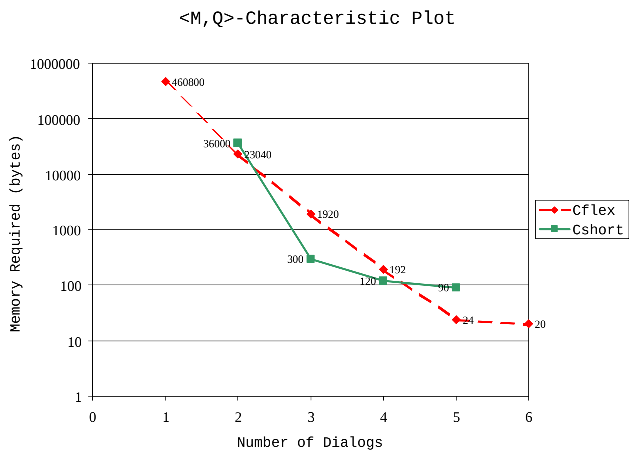

Figure 8: Sample $\langle m, q\rangle$-*characterisation* plots for *Cflex* and *Cshort*. Due to reorganisational constraints, *Cshort* has a minimum of two questions. *Cflex* is more flexible in that it can support devices with lower memories.

PROOF OUTLINE. Every vertex in T^r_m can be created by merging several vertices in T_a. Therefore, each vertex in T^r_m corresponds to a unique subtree in T_a. The *unfolding* operation identifies this subtree, and attempts to *shorten* it as much as possible to minimise alteration. Since each vertex corresponds to a unique subtree, the order of replacing the subtrees is inconsequential. □

3.3 Call-flow Characterisation

Given a call-flow C, the above algorithms can be run with various values of memory size $m_i, 1 \leq i \leq n$ and their corresponding minimum number of questions obtained. This gives us a *device-independent* characterisation of C. Since these $\langle m_i, q_i\rangle$-pairs are unique for a given call-flow, they can be thought of as a *reorganisational signature* of the call-flow. We call this signature an $\langle m, q\rangle$-*characterisation* of C. From a practical perspective, the $\langle m, q\rangle$-*characterisation* of C provides a means for comparing two call-flows that essentially (semantically) perform the same task, that of doing airline reservation, for example, and traces the memory requirements of each. This is important in call-flow design.

The $\langle m, q\rangle$-*characterisation* function of C is typically a decaying function – a composition of lines with negative, decreasing slopes. Consider a sequential call-flow L of n dialogs where each dialog i requires memory m_i, a single question requires $\Pi_1^n m_i$. This is the largest value of the function. The smallest value is $max\ m_i$. When all the numbers are the same, this function reduces to an exponential function on m_1. In the most general case, this function is similar to the *falling factorial* function, except that the the numbers are not necessarily consecutive, so the slope of the curve continues to decrease faster than the *falling factorial* function. In the case of tree-type call-flows, since the numbers get added rather than multiplied, the effect is less pronounced.

Figure 8 shows a comparison of the $\langle m, q\rangle$-*characteristics* of two imaginary call-flows, *Cflex* and *Cshort*. Note that both call-flows are semantically equivalent in that they perform the same task (for example, airline reservation) but were designed with different assumptions and considerations in mind. The choice of the call-flow could depend on a number of factors. For example, if the designer expects client devices (with less than 90 bytes) to access the application, *Cflex* is preferable. However, if the designer is concerned that he does not want to ask more than 3 questions, then *Cshort* accomplishes this with lesser memory.

Table 1: Memory requirement for the different grammar sizes

Grammar size	Memory required (bytes)
1	47916
116	47960
280	48080
370	48052
960	48412
1440	48644
3000	53356
4000	59712
5000	60532
7000	61888
9600	62860
10670	63060

4. EXPERIMENTS

In this section, we present the experiments conducted to validate our assumptions and to evaluate the performance of our proposed algorithms, RESEQUENCE and MASQ, with a sample sequential call-flow. In section 4.1, we illustrate the effect of the size of a grammar on the memory requirements of a speech recognition system. We use a large vocabulary continuous speech system to decode a single utterance with varying grammar size. We show the outcome of running RESEQUENCE on a sample sequential call-flow of an Airline Reservation system in section 4.2. Section 4.3 describes the implications of the RESEQUENCE algorithm with distance measure on the same sample call-flow.

4.1 Effect of Grammar Size on Memory Requirement

We used a large vocabulary English speech recognition system to decode a speech utterance. We used 24-dimensional Mel Frequency Cepstral Coefficients (MFCC) as the feature vector of the speech data. To capture the dynamics of the speech signal, 4 previous and 4 succeeding MFCC vectors were concatenated to the current MFCC vector and Linear Discriminant Analysis (LDA) was applied on the concatenated vector to reduce the dimensionality of the feature vector from 24×9 to 60 dimensions. The vectors so obtained were used to model the output distribution of Hidden Markov Models (HMM). The acoustic models were trained over 34 hours of speech data collected from more than 170 speakers. The utterance comprised of a single word. The grammars used for decoding were isolated words. The size of grammar therefore reflects the vocabulary of the recognition system. The decoding was performed for the same utterance, but with varying grammar sizes. Table 1 shows the memory required to perform decoding on a AIX machine with 2GB of RAM. The experiments were performed on on a 450 MHz Quad processor. These numbers may vary depending upon the particular implementation of the speech recognition system and the hardware, but with an increase in grammar size, an increase in memory requirement is expected.

The memory requirement of 47916 bytes for decoding the utterance against a one word grammar can be interpreted as the footprint that is required by the non-grammar specific portion of the speech recognition system. This memory is used to compute and store the cepstrum features from the speech signal, and to store the HMM

Table 2: Memory capacity of various devices.

Index	Device	Memory Capacity
A	Compaq Ipaq H3970	64 MB
B	Mitac Mio 338	40 MB
C	Sony Clie PEG-NX70V	16 MB
D	Nokia 6600	3 MB
E	Nokia 7650	1.4 MB
F	Nokia 6585	512 KB
G	Nokia 3100	210 KB

parameters. The additional memory requirement with the increase of grammar size is highlighted when the same front-end processing and the same HMM is used for varying sizes of memory. Therefore the experiment validates that increasing the grammar size results in an increased requirement for decoding the same speech utterance.

The memory required by a speech recognition system has two components: one that varies with the increase in vocabulary size, and the other that is constant. The latter part is due to the memory by the acoustic model and the code store of the speech recognition system.To estimate the memory required by the constant portion of a speech recognition system, we use the product brief of Sensory® Fluent Speech™ software [11]. The footprint of the Fluent Speech™ recognition engine as obtained from their SDK [10] document is 750 Kb for a vocabulary of 500 words. The memory required for additional words is 250 bytes per word [11]. Assuming a linear relationship between the vocabulary of the system and its memory requirement, we estimate the memory footprint of the constant portion of a speech recognition system as $750 - 500 * 0.25 = 625$ Kb. Next, we estimate the memory requirement for a given call-flow. For this, we assume that the vocabulary size of the speech recognition system for a particular grammar is equal to the number of choices in that grammar.We do not handle the complications in designing grammars that arise due to the natural language options.

Table 2 shows the memory capacity of various handheld devices including mobile phones. The numbers vary from as low as 210 KB to as high as 64 MB. Each of these devices has the processing power to be able to run a speech recognition system locally. Such variation in memory capacities provides a justification for device-specific call-flow reorganisation.

4.1.1 Example: Airline Reservation Call-flow

Figure 9(a) shows the atomic call-flow of an Airline Reservation system. This is a sequential call-flow which means that a following question asked by the system is independent of the user's response to a previous one. Each atomic dialog is of the form where the system asks the user a question such as "What is your 'departure city'?", and collects atomic information when the user responds with a valid choice (a valid city name in this case). For each dialog the grammar corresponding to the answer of the question is loaded for speech recognition. The memory (in KB) required by the speech recognition system to process the grammar is shown in $< >$ against each dialog in figure 9. Once the user's response is recognised, the system asks the next question. It requires 13 fields to be provided by the user. The minimum memory required for this atomic call-flow over and above the recognition engine footprint is 29 Kb, which corresponds to the largest grammar ('Departure city'/'Arrival city'). Table 2 shows the memory requirement of the atomic grammars of the call-flow.

The reorganisation constraints for the call-flow are indicated by the dotted lines in figure 9(a). There are five reorganisation constraints as shown in the figure. The constraints are as follows: Do not merge 'Departure city' with 'Arrival city', 'Arrival city' with 'Day of journey', 'Month of journey' with 'Airline name', 'Number of seats' with 'Credit card type' and 'Credit card expiry year' with 'Contact number'.

4.2 RESEQUENCE

The atomic call-flow (figure 9(a)) along with the memory constraint of the particular device and the reorganisation constraints are inputs to the RESEQUENCE algorithm. The algorithm outputs the optimal dialog call-flow for the device. We show in figure 9(b) and 9(c) two such call-flows with different q values. Figure 9(b) shows the call-flow with $q = 9$. The minimum memory required for this call-flow over and above the recognition engine footprint is 70 Kb, which corresponds to the grammar 'Airline name & Flight number'. This optimal call-flow is obtained by merging the following questions : ('airline' & 'flight number'), ('class of reservation' & 'number of seats'), ('credit card type' & 'credit card number'), ('credit card expiry month' & 'credit card expiry year').

The dialog call-flow in figure 9(c) is the call-flow with minimum possible questions, $q = 6$, taking into consideration the reorganisation constraints. The minimum memory required for this call-flow over and above the recognition engine footprint is 2400 Kb. This optimal call-flow is obtained by merging the following questions : ('day of journey' & 'month of journey'), ('airline' & 'flight number'), ('class of reservation' & 'number of seats'), ('credit card type', 'credit card number', 'credit card expiry month' & 'credit card expiry year').

The $\langle m, q \rangle$ characteristics of the above call-flow is shown in Figure 11. The x-axis of this bar chart reflects to the number of questions in the dialog call-flow and the y-axis shows the memory that would be required to execute the particular call-flow[5]. Note that each bar in the chart corresponds to a unique call-flow. The value on the y-axis refers to the memory required to execute the largest grammar in the respective call-flow. The plot has been generated by running the RESEQUENCE algorithm on the call-flow mentioned in Figure 9(a) by varying m and finding the corresponding q values.

4.3 MASQ

Figure 10(a) shows an ideal call-flow corresponding to the call-flow shown in figure 9(a). The ideal call-flow, atomic call-flow, reorganisation constraints and the device's memory resources form an input to MASQ. The output of the algorithm is the optimal call-flow with minimum distance between the output and the ideal call-flow. Figure 10(b) shows a call-flow for $q = 12$. The minimum memory required for this call-flow over and above the recognition engine footprint is 29 Kb.

The implication of the distance between two call-flows can be observed by comparing the output of RESEQUENCE and MASQ. With reference to the atomic call-flow described in figure 9(a), for $q = 12$ the output of the two algorithms would be different. RESEQUENCE can output a merged grammar corresponding to either 'Flight number' with 'Class of reservation' or 'Class of reservation' with 'Number of seats'. The output is such because the memory requirement for both the merged grammars is same. So RESEQUENCE algorithm can arbitrarily pick one of them. However in MASQ, the output call-flow would have a merged grammar of 'Class of reservation' and 'Number of seats'. This is because the resulting call-flow is at the least distance from the ideal reference call-flow 10(b).

[5]This does not include the memory required by the recognition engine.

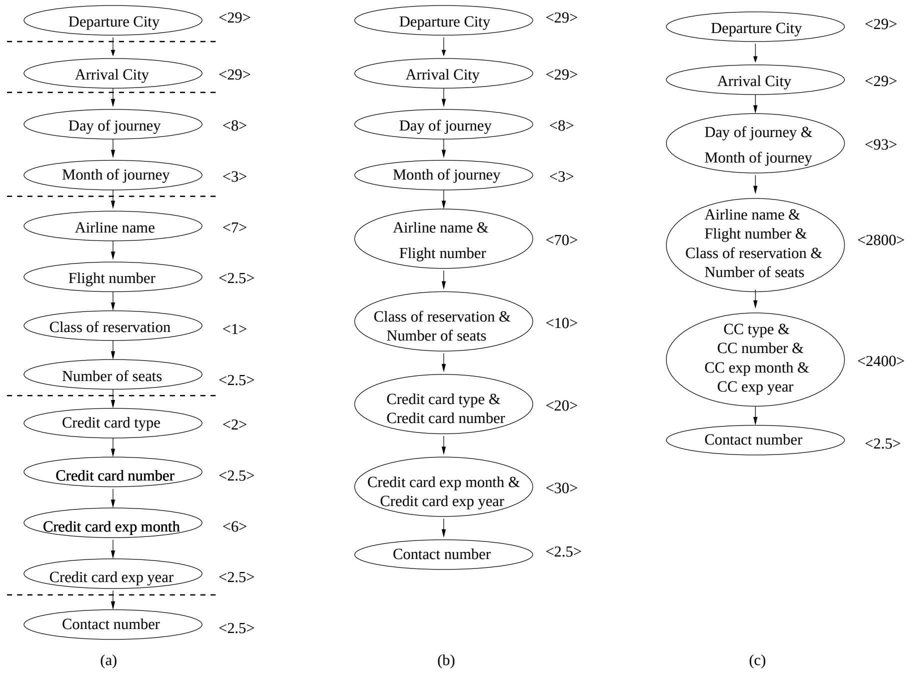
Departure City
<29>
Arrival City
<29>
Day of journey
<8>
Month of journey
<3>
Airline name
<7>
Flight number
<2.5>
Class of reservation
<1>
Number of seats
<2.5>
Credit card type
<2>
Credit card number
<2.5>
Credit card exp month
<6>
Credit card exp year
<2.5>
Contact number
<2.5>
(a)
Airline name & Flight number
<70>
Class of reservation & Number of seats
<10>
Credit card type & Credit card number
<20>
Credit card exp month & Credit card exp year
<30>
(b)
Day of journey & Month of journey
<93>
Airline name & Flight number & Class of reservation & Number of seats
<2800>
CC type & CC number & CC exp month & CC exp year
<2400>
(c)

Figure 9: (a)An example atomic call-flow. (b)Output of the RESEQUENCE algorithm with q=10. (c)Output of the RESEQUENCE algorithm with q=6. The memory (in KB) required by the speech recognition system to process the grammar is shown in $< >$ against each dialog.

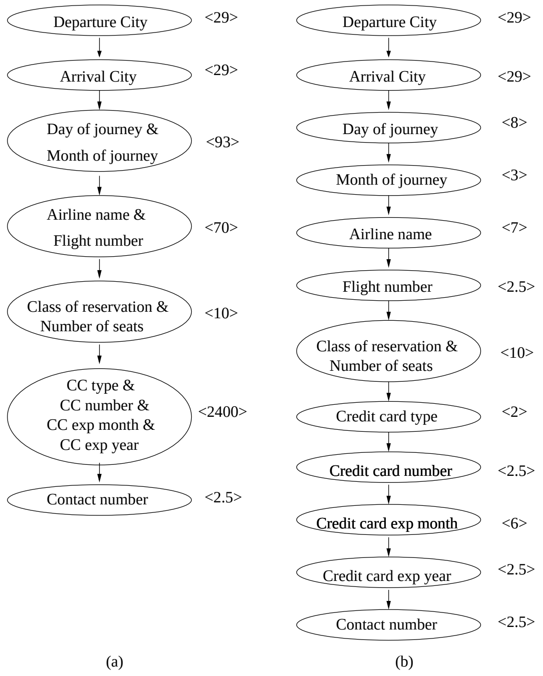

Figure 10: (a)An ideal reference call-flow, (b) Output of the MASQ for q=12.

There are other parts to the dialog such as the welcome message and goodbye greeting. These make the dialog more human-friendly, but have no effect on the execution of the algorithms.

5. SUMMARY AND FUTURE WORK

Ubiquitous information access via multiple modalities and devices defines a growing need. Transforming this need into a reality will require enormous efforts and insights. The development of specifications such as VoiceXML coupled with improvement in speech recognition and speech synthesis have made speech a viable alternative today.

Multi-device access poses a further challenge. Speech applications need to be adapted to different devices. We investigated the problem of dialog call-flow reorganisation for pervasive devices with memory size restrictions. The crux of the reorganisation lies in altering the memory requirements of the underlying grammar. We achieve this by merging atomic grammars while minimising the number of questions, thus accounting for one aspect of usability. We presented optimal algorithms to achieve this.

Our work provides a key component around which tools can be developed for automatic adaptation of speech dialogs for multiple devices. We have taken a small step in that direction by identifying some real requirements, formalising a subproblem and articulating some ideas that might hint towards a possible solution.

As is the case with all user interface technologies, usability is a critical factor that measures the goodness of the automatically designed call-flows. Future efforts in this direction can be to formulate a constrained optimisation problem by integrating the existing dialog evaluation methods [13] with the device specific optimisation techniques mentioned in this paper.

6. REFERENCES

[1] G. Banavar, L. D. Bergman, Y. Gaeremynk, D. Soroker and

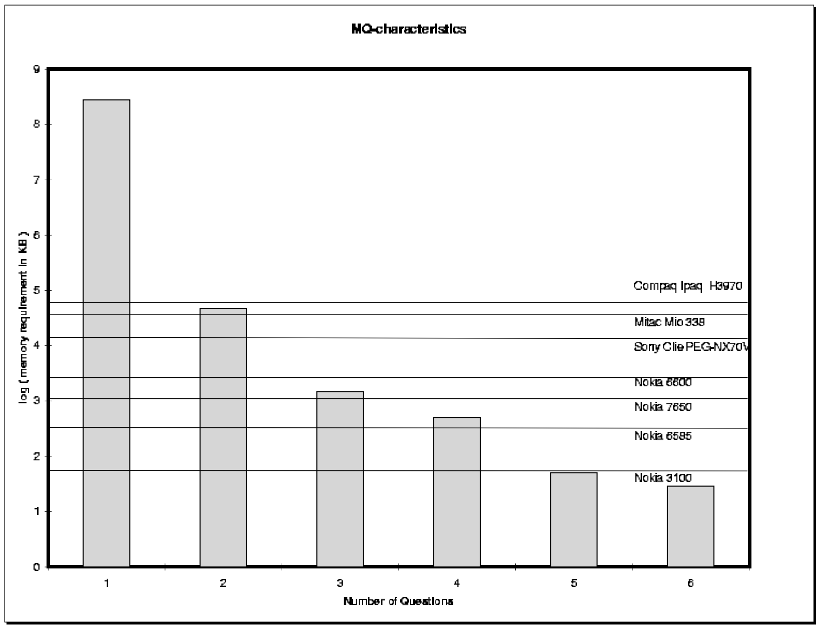

Figure 11: The $\langle m, q\rangle$-characterisation of the dialog call-flow shown in Figure 9.

J. Sussman, "Tooling and System Support for Authoring Multi-Device Applications," Journal of Systems and Software, Vol. 69, Issue 3, January 2004, pp. 227-242.

[2] R.E. Donovan and E.M. Eide,"The IBM Trainable Speech Synthesis System," International Conference on Spoken Language Processing, Sydney, 1998.

[3] A. Jameson, "Adapting to the user's time and working memory limitations: New directions of research," ABIS-98, FORWISS.

[4] D. Litman and S. Pan, "Emperically Evaluating an Adaptable Spoken Dialogue System," International Conference on User Modeling, Banff, Canada, 1999.

[5] P. Heisterkamp, W. Minker, U. Haiber and S. Scheible, "Intelligent Dialog Overcomes Speech Technology Limitations: The SENECa Example," International Conference on Intelligent User Interfaces, Miami, Florida, January 12-15, 2003, pp. 267–269.

[6] E. Levin, R. Pieraccini and W. Eckert, "A Stochastic Model of Human-Machine Interaction for Learning Dialog Strategies," IEEE Transactions on Speech and Audio Processing, 8(1), January 2000.

[7] M. Padmanabhan, G. Saon, G. Zweig, J. Huang, B. Kingsbury and L. Mangu, "Evolution of the performance of automatic speech recognition algorithms in transcribing conversational telephone speech," Instrumentation and Measurement Technology Conference, 2001. IMTC 2001. Proceedings of the 18th IEEE, Vol. 3, 21-23 May 2001, pp. 1926–1931.

[8] B. Peskin, L. Gillick, N. Liberman, M. Newman, P. van Mulbregt and S. Wegmann, "Progress in recognizing conversational telephone speech," ICASSP-97, 21-24 April 1997, pp. 1811–1814.

[9] G. N. Ramaswamy and P. S. Gopalakrishnan, "Compression of acoustic features for speech recognition in network environments," ICASSP98, Vol 2, pp 977–980.

[10] http://www.sensoryinc.com/html/support/docs/80-0216-A.pdf

[11] http://www.sensoryinc.com/html/support/docs/80-0193-0.pdf

[12] M. Walker, J. Boland and C. Kamm "The Utility of Elapsed Time as a Usability Metric for Spoken Dialog Systems," ASRU, Colorado, 1999.

[13] M. Walker, D. Litman, C. Kamm and A. Abella, "Paradise: A Framework for Evaluating Spoken Dialogue Agents," Proceedings of Annual Meeting of the Association for Computational Linguistics, Madrid, Spain, 1997.