Who's Asking For Help? A Bayesian Approach to Intelligent Assistance

Bowen Hui
Department of Computer Science
University of Toronto
bowen@cs.utoronto.ca

Craig Boutilier
Department of Computer Science
University of Toronto
cebly@cs.utoronto.ca

ABSTRACT

Automated software customization is drawing increasing attention as a means to help users deal with the scope, complexity, potential intrusiveness, and ever-changing nature of modern software. The ability to automatically customize functionality, interfaces, and advice to specific users is made more difficult by the uncertainty about the needs of specific individuals and their preferences for interaction. Following recent probabilistic techniques in user modeling, we model our user with a dynamic Bayesian network (DBN) and propose to explicitly infer the "user's type" — a composite of personality and affect variables — in real time. We design the system to reason about the impact of its actions given the user's current attitudes. To illustrate the benefits of this approach, we describe a DBN model for a text-editing help task. We show, through simulations, that user types can be inferred quickly, and that a myopic policy offers considerable benefit by adapting to both different types and changing attitudes. We then develop a more realistic user model, using behavioural data from 45 users to learn model parameters and the topology of our proposed user types. With the new model, we conduct a usability experiment with 4 users and 4 different policies. These experiments, while preliminary, show encouraging results for our adaptive policy.

Categories and Subject Descriptors: I.2.11 [Artificial Intelligence]: Intelligent agents H.5 [Information Interfaces and Presentation]: Miscellaneous

General Terms: Human Factors

Keywords: User modeling, dynamic Bayesian networks, intelligent assistance

1. INTRODUCTION

Online software customization has become increasingly important as users are faced with larger, more complex applications. For a variety of reasons, software must be tailored to specific individuals and circumstances [18]. For example, adaptive interfaces are critical as different users may require different functionality from multi-purpose software [5], prefer different modes of interaction, or use software on a variety of hardware devices [10]. Because of software complexity, online and automated help systems are becoming increasingly prevalent to help users identify and master different software functions [16]. Such systems should ideally adapt the help they provide and the decision to interrupt [15] to account for specific user preferences.

One of the difficulties facing developers of adaptive software, interfaces, and help systems is the uncertainty associated with assessing the needs of a specific user. While hard-coded rules offer some benefits, it is becoming apparent that probabilistic assessment of a user's needs based on observed behaviour offers considerable advantages [16, 1, 11]. Such approaches employ detailed models (either handcrafted or partially learned) and often multimodal inputs to infer the user's goals in the current system environment. Few of these approaches model user features explicitly (though some exceptions exist, e.g., systems in which user features are taken as input [6, 19, 3]). It is rarer still to learn such features. One exception is the work of Zhou and Conati [23] which infers a user's emotional states; but it is unclear how this information impacts the system's actions. We argue that it is more natural to model human-computer interaction as a *sequential* stochastic process where the *user* moves from state to state. Here, states reflect the user's attitudes and abilities (cognitive, motor, etc.) as well as the system's environment. In this way, our method infers and adapts to the user's current attitudes, and learns an on-going user profile that may be transferred to other applications.

In addition, complex tradeoffs must be assessed when deciding if and when to offer help to a user, hide a specific function, etc. For example, deciding to offer help must balance the uncertain assessment as to whether help is needed, the costs of unwanted interruption, the benefits of providing the right type of help, and the costs of providing the wrong type of help or of doing nothing when help is needed. In a number of settings, decision-theoretic models have been adopted for precisely this reason [16, 1, 15, 3, 19, 10, 9, 2], allowing a system to make the right decision based on such decision-theoretic tradeoffs. We adopt this general perspective, but tailor our approach so that decisions are influenced by the system's beliefs about the (generally evolving) user state. We achieve this via online *belief state monitoring*, which we show to be tractable in the system prototype.

Our aim is the development of systems that can actively monitor user behaviour and tailor an interface, help system, or functionality to the needs of that user. Furthermore, we want systems that construct models of a user over time to support this customization, and whose actions are influenced by the need to develop accurate user models. We focus on an automated help system for a text-editing task (with an eye toward users with mild cognitive or physical impairments), but the general principles apply more broadly.

IUI'06 January 29–February 1, 2006, Sydney, Australia.

More precisely, we develop a generic model of *static user type* and *transient user state* in which both the type and the state are inferred (or learned) over time based on observations of user behaviour. We model the dynamics of user state and the interaction with a help system using a dynamic Bayesian network, and the relative benefits of various types of help (and their interaction with user state) using a generalized additive utility model. The probabilistically estimated user state is then used determine the expected utility of a specific course of action (various forms of help or lack of help) at any point in time. This generic model is elaborated in Section 2. We instantiate this model in a specific text-editing task in Section 3, with assistance for users with mild cognitive or physical impairments in mind. However, the general principles illustrated in this task carry over to any form of automated software assistance. We discuss simulation results in Section 4, a protocol for learning model parameters in Section 5, and the results of a preliminary user study in Section 6. While the user study suggests certain problems with the prototype implementation, the qualitative results are quite encouraging and do suggest that this general decision-theoretic approach to assistance we propose is indeed useful.

2. A GENERIC USER MODEL

We begin by proposing a generic model that allows an automated assistant to learn about its user.

2.1 User State and User Types

First we consider the factors that influence whether a user accepts help from an automated assistant, and the value of such assistance, as shown in Figure 1 (left). Whether a user accepts automated help depends on the quality of the assistance (QUAL) as well as the user's tendency to work independently (TI) and the amount of attention that is directed toward considering help (CONS). For example, a user who is highly independent may not consider or accept help even if it is perfect. The degree to which a user might consider help depends on the user's current attitudes toward the automated agent and general personality traits while working in a computing setting. Relevant user attitudes include those directed toward the computing environment, such as frustration (F), and those toward the immediate task, such as neediness (N). Relevant personality traits include the user's tendency to get distracted (TD) and tendency to work independently (TI) on a computer. These influences are illustrated in Figure 1 (right). Other factors can be modeled similarly.

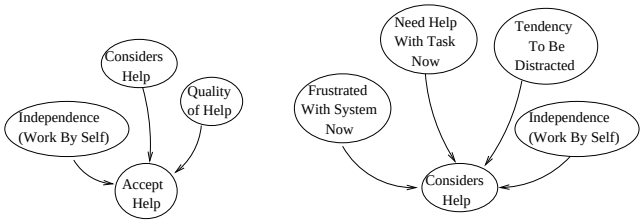

Figure 1: Influential factors. Left: Causes for accepting help. Right: Causes for considering help.

Together, the variables $\{F, N, TD, TI\}$ make up the user's *state*. As we will see, these are sufficient to predict the probability of specific user behaviours (including accepting help) and how costly or rewarding a user perceives his experience with the automated system at any point in time. Consider the follow examples: someone who is easily distracted may find automated assistance costly because it prevents the user from completing the task; someone who currently needs help with a difficult task may benefit greatly from partial suggestions that helps the user identify the next steps; someone who is generally dependent may not mind receiving imperfect suggestions as much as someone who is highly independent; someone who is frustrated with the system now is likely to become more frustrated with further interruptions and suggestions. We discuss the precise structure of the reward and cost functions below.

Variables *TD* and *TI* are *static*, reflecting specific user traits that do not change over time.[1] In contrast, *F* and *N* are *transient*, reflecting user attitudes that can change, often frequently, during a specific session. How these transient variables evolve can also be modeled by assuming additional static user traits. For this purpose, we propose latent variables *TF* and *TN*, representing the user's *tendencies* toward frustration and neediness in the application. These influence the (stochastic) evolution of *F* and *N*. We define a user's *type* to be the state of all static user traits: $\{TF, TN, TD, TI\}$.

In our prototype application, these user variables are discrete, with variables *F*, *N*, *TD* and *TI* having 3 values each, and *TN* and *TD* having 2 values each. $F = 1$ denotes that the user is not frustrated, $F = 2$ the user is somewhat frustrated, and $F = 3$ the user is very frustrated. Other variables are defined similarly. As a result, there are 81 user states and 36 user types.

2.2 Model Structure and Dynamics

Since the user state is partially observable, the system must maintain a *probability distribution*, or belief state $BEL(F, N, TD, TI)$ (BEL for short), over user states given all past observations of user behaviour (reflecting the relative likelihood that the user is in a particular state). Based on the current belief state, the system reasons about the rewards and costs of its actions in order to make an appropriate decision, and updates its beliefs after each user observation. A user's type (over traits *TF*, *TN*, *TD*, *TI*), despite having a fixed value for a specific user, is not known *a priori* to the system and thus must also be estimated probabilistically.

The causal relationships in Figure 1 form the basis of our model. In addition, the availability of automated assistance (HELP) affects when the user can consider suggestions. Our model incorporates additional system variables (SYS) and user observations (OBS). An example of a system variable is the status of an interface widget that allows the user to directly manipulate its settings. User observations should be abstracted at a behavioural level, and useful for inferring the user's state. Since these observations are domain-specific, we leave further discussion to Section 3.1.

At a given point in time, the system observes the user's action, infers the user's current state, and decides whether to offer help at the next time step. Naturally, certain variable values may persist over time, or influence the values of other variables at the next point in time. For example, a user may be frustrated now, but over time, the frustration level will decrease if nothing else aggravates him. To model these temporal characteristics, we adopt a two-stage dynamic Bayesian network (DBN) model [7], as shown in Figure 2. In this model, variables F, N, and CONS have temporal dependencies on their counterparts in the future, and the values of the user type variables persist over time. This model allows the system to learn the user's type through behavioural observations.

Formally, a two-stage DBN models a joint distribution over a set of n random variables $\mathbf{X} = \{X_1, ..., X_n\}$ at time $t-1$ and t. We denote the parameterization as θ, which specifies the set of conditional probability distributions, $Pr(X_i|Pa_i)$ for each X_i and its parents Pa_i. In particular, $\theta_{ijk} = Pr(X_i = x_i^k|Pa_i = pa_i^j)$ for

[1]Naturally, these can change over certain time scales, but we take these to be static at least over the time frame associated with a reasonably small series of application sessions.

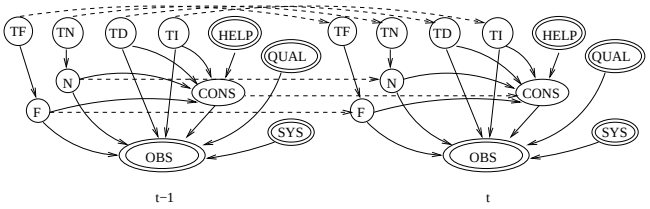

Figure 2: A two time-step DBN user model. Solid arcs indicate intra-temporal links while dashed lines indicate inter-temporal links. Observations are drawn with double lines.

the k^{th} value of X_i and the j^{th} parent configuration. We discuss these parameters in more detail in the next section.

The parameters of a DBN are defined by a prior distribution at time $t = 0$, a transition function, and an observation model. As an initial step, we handcrafted these parameters using expert domain knowledge. The transition function for the user types is the identity function, since these variables represent persistent traits of a user. The transition functions $Pr(F_t|F_{t-1}, TF_t)$ and $Pr(N_t|N_{t-1}, TN_t)$ capture frustration and neediness patterns and how they evolve. The transition function for $CONS_t$ and the observation model for OBS_t are more complicated due to the size of the distributions. Here, we exploit their common substructure. For example, if help is not available, the user is not considering it, $Pr(CONS_t = not|HELP_t = none) = 1.0$. If help is available and the user was considering it ($CONS_{t-1} = yes$), then the probability of the user considering help now is defined as $Pr(CONS_t|F_t, N_t, HELP_t)$, which is independent of TD_t and TI_t. The intuition is that whether one will (dis)continue to consider help depends on changes in the levels of frustration or neediness. On the other hand, if the user was not considering help already, then the current consideration level will depend on the difficulty of the task and the user's tendency to work alone, $Pr(CONS_t|N_t, TI_t, HELP_t)$.

Exact inference in DBNs is done via the *clique tree* algorithm [17]. The performance of this algorithm depends on the size of the cliques which are created based on the dependencies in the DBN. In the context of user modeling, we are interested in monitoring the system's belief distribution over the user's state over time, given past observations: $Pr(BEL_t|OBS_{1:t})$. Let X_t denote the clique consisting of elements BEL_t, TF_t, TN_t, and $CONS_t$. Then $Pr(BEL_t|OBS_{1:t}) = \sum_{TF_t, TN_t, CONS_t} Pr(X_t|OBS_{1:t})$ which is proportional to $\sum_{TF_t, TN_t, CONS_t} Pr(OBS_t|X_t)Pr(X_t|OBS_{1:t-1})$. This equation corresponds to a *rollup* step in the inference algorithm. In Section 4, we discuss simulation results that gauge the speed and accuracy of this process.

We are also interested in predicting the likelihood of a user accepting help given its quality, the system environment, and past evidence: $Pr(OBS_{t+1} = acc|HELP_{t+1}, QUAL_{t+1}, SYS_{t+1}|OBS_{1:t})$. This term can also be computed readily using the clique tree algorithm and is used in the system's decision making policy, which is described in Section 3.3.

2.3 Reward Function

In modeling a wide range of user types, we must consider multiple conflicting objectives: for general users, a level of independent functioning is considered desirable, so there is some cost to help; there is benefit of providing the right help when needed or desired; there is a cost to providing incorrect suggestions, or suggestions when not needed or desired. Furthermore, the system should customize the degree of help based on its beliefs about the user's current attitudes.

To evaluate automated help, we define a reward and cost function that incorporate user preferences toward automated assistance. The reward function depends on the user state and the quality of the suggestion, $R(F, N, TD, TI, QUAL)$ and is decomposed as follows: $R(F, TI, QUAL) + R(N, TI, QUAL) + R(TD, QUAL)$. This generalized additive decomposition reflects the assumption that the overall perceived value of help (of some specified quality) can be determined by independent contributions given the current levels of frustration and neediness (each of these conditioned on degree of independence) and degree of distractibility. The cost of interrupting the user is defined as $C(F, N, TD, TI)$, irrespective of the quality of the automated help. We assume additive independence of the cost function: $C(F) + C(N) + C(TD) + C(TI)$. We normalize the range of the rewards and costs to be in [-40,40].

3. TEXT-EDITING ASSISTANCE

In order to infer a user's state, we need to identify observations that correlate with those states. Therefore, the detailed structure of the model must be domain-specific. We chose a text editor as a testbed application because it is familiar to many computer users and its functions are common to other communication software such as email and online chat. Furthermore, people with vocabulary and motor disadvantages often find that word processing and word prediction software allow them to concentrate on the quality of writing and give them a sense of authorship [14]. Within the editor, word prediction is treated as automated help. The architecture is presented in Figure 3. Unlike other word prediction software, our system will not offer suggestions whenever a letter is typed. Rather, it learns the user's traits and needs and make suggestions only when it believes that the user can benefit from them. This methodology is generalizable to more complex software and tasks.

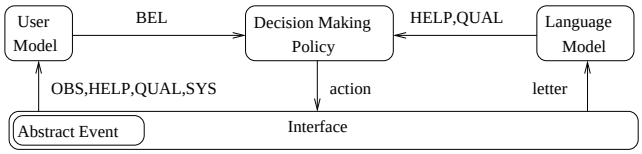

Figure 3: Overall system architecture.

3.1 Deriving Fully Observable Variables

In a typical computing environment, keyboard and mouse events are the source of fully observable variables. We abstract these events into *behavioural* patterns that correlate with user characteristics. The resulting set of observations modeled in the variable *OBS* can be roughly categorized according to the various user state characteristics with which they are correlated:

- **Frustration:** continuously pressing a key down, moving the mouse back and forth quickly, jamming into the keyboard, multiple fast mouse clicks, explicitly indicating a need for fewer suggestions
- **Neediness:** erasing many characters, browsing (surfing menus, switching applications) for help, pausing
- **Distractibility:** browsing (surfing menus, switching applications) due to distraction, pausing
- **Independence:** explicitly indicating a need for more or fewer suggestions, accepting help/suggestions (as a function of quality)

Note that browsing and pausing are common to both neediness and distractibility, which is consistent with other proposed models [16]. This ambiguity creates additional uncertainty that the system needs to manage and further suggests the importance of a probabilistic model that account for multiple "causes" for observed behaviour. Other user behaviours include responses to automated suggestions, such as accepting help (acc), hovering over the suggestion box (hh), and pausing when suggestions are present (hp). We also created a slider widget, SDR, that allows the user to explicitly indicate whether more or fewer suggestions are desired.

Under this design, the system has 2 actions — to offer a set of completion words (POP), or to remain passive (¬POP). Together, there are 972 hidden states and 420 observations, yielding a total of 408,240 system states. The DBN model allows us to keep the representation compact in terms of the local distributions, rather than using a *flat* state representation (whose size is exponentially larger).

3.2 Language Model

The word prediction component is treated as a plug-in module in the system's reasoning process. This module takes as input the previously typed word, w_{t-1} and the current prefix, w_{prefix}. As output, it returns a set of suggestions with a quality estimate. This quality value is important because it directly impacts whether a user will accept automated assistance. Furthermore, in a word prediction domain, the quality of the predictions vary widely depending on the prediction algorithm used. These factors have a strong influence on the system's decision whether to offer help as we will see below.

Standard word prediction software makes use of collocation statistics such as *n-gram probabilities* [22]. In particular, for $n = 2$, a *bigram* probability is defined as $Pr(w_t|w_{t-1})$. (In a prediction task, w_t must be consistent with w_{prefix}.) Our system also adopts a bigram model, which is trained on 40% of the 100 million word British National Corpus (BNC). The system maintains the top 20,000 bigrams and 20,000 unigrams with backoff weights in its lexicon at runtime. In addition to using the bigram probabilities, we want the suggestion feature to offer completion words that are *different* from each other. In other words, we want the completions to cover a larger probability mass. For example, with $w_{t-1} =$ "the" and $w_{prefix} =$ "nu", a bigram model may offer suggestions "number", "numbers", and "nuclear" even though "number" and "numbers" only differ by one letter. Therefore, we propose a similarity metric that captures the *expected savings* a word provides to the user.

At a given point in time, there is a set of $\{c_1, ..., c_K\}$ words that are plausible (i.e., non-zero probability) completions given w_{t-1} and w_{prefix}. Each c_k has an associated bigram probability, p_k. To attribute a utility measure to a suggestion, we first define its utility with respect to a true word s defining $U(c_k|s)$ to be the number of identical prefix characters less the number of characters erased less the number of characters added to change c_k into s [9]. For example, U("are"|"all") $= 1 - 2 - 2 = -3$, while U("apples"|"apple") $= 5 - 1 - 0 = +4$. Given $U(c_k|s)$, we define the *expected savings* of c_k as $ES(c_k) = \sum_{i=1}^{K} U(c_k|c_i)p_i$, where p_i is c_i's bigram probability. We define the *joint* expected savings $JES(c_1, ..., c_J) = \sum_{i=1}^{K} argmax_{c_j} U(c_j|c_i)p_i$ for a suggestion with J words. The intuition is that, for any true c_i, the user will accept the suggestion (among the J) offering maximum savings. In the example with $w_{t-1} =$ "the" and $w_{prefix} =$ "nu", the JES model chooses the suggestions: "number", "nuclear", and "nurses".

Unfortunately, when $J \geq 2$, the number of comparisons increases exponentially. We propose a greedy implementation for our JES model. First, among K words, choose $s_1 = argmax_{c_k} ES(c_k)$. With $K - 1$ words left, choose the second best completion with respect to s_1; that is, $s_2 = argmax_{c_k} JES(s_1, c_k)$; and so on. This greedy approach results in $O(J - 1 \cdot K)$ comparisons. The estimated quality of a suggestion is simply its joint expected savings.

Table 1 shows a comparison of these algorithms through experiments implemented in Matlab and ran in Linux with 3.60G Hz CPU, with $K = 40$ and $J = 3$. We use bigrams as the baseline comparison by taking the J most probable words, and we show the speed performance of using JES for word prediction, implemented both greedily and by full enumeration. We ran the three algorithms through a text of length 11,718 characters (with 7917 word prediction opportunities). We see that the average and maximum times for the bigram and greedy techniques are similar, while the enumerative method is too slow for an online task. We kept track of the number of correct predictions made (Exact), the number of predictions that contained a substring of the true word (Substr), and the actual character savings (Util). The JES greedy implementation scores almost as well as the bigram model on correct predictions, but this is not our main concern. Critically, the JES model provides much greater utility with respect to character savings. It is also significantly faster than full enumeration but still offers acceptable performance with respect to utility (note that Enum provides optimal suggestions). Results from the usability experiments in Section 6 also suggest that the expected savings metric is more helpful for users.

Table 1: Comparison of prediction techniques

Method	Avg (s)	Max (s)	Exact	Substr	Util
Bigrams	0.1485	0.9693	3629	5674	9151
Greedy	0.1743	1.0285	3578	5706	9740
Enum	0.7110	3.5008	3708	5806	10169

3.3 Decision Policy

The decision problem faced by the help system is characterized by considerable uncertainty. Obviously, the word a specific user is typing cannot be predicted with certainty, though the language model allows us to quantify this probabilistically and rather precisely. More importantly, whether a user could benefit from the system's help, or desires such help, cannot be assessed with certainty either. Our model is designed to (probabilistically) predict whether a user needs or wants help based on past observed user behaviour.

We define a myopic policy that models the uncertainty and systematically trades off the conflicting objectives as follows.[2] At each time step, the system takes an action and the user can either accept it (*OBS* $= acc$) or not (*OBS* $= \neg acc$). Considering these possible outcomes, the *expected utility* of an action is $EU(POP) = EU(POP|acc)Pr(acc) + EU(POP|\neg acc)Pr(\neg acc)$. If the user accepts a suggestion, the system will "receive a reward" reflecting the net benefit of the suggestion (incorporating any costs of interruption, etc.). Of course, the system can only compute the *expected* reward since the user state is not fully known. Thus, we define $EU(POP|acc) =$

$$\sum_{F,N,TD,TI} R(F, N, TD, TI, QUAL)BEL(F, N, TD, TI)$$

[2]Ultimately, we expect much better performance taking sequentially informed decisions by solving the partially observable Markov decision process (POMDP) induced by this model. We discuss this in the concluding section.

On the other hand, if the user rejects the suggestion, the system will receive a penalty, again, in expectation given the user's type. We define $EU(POP|\neg acc) =$

$$\sum_{F,N,TD,TI} C(F,N,TD,TI)BEL(F,N,TD,TI)$$

We predict the value of making a suggestion by taking the expected value of POP with respect to the probability of acceptance. The overall system policy is to take the action with the maximum expected utility (MEU): *pop up a suggestion if* $EU(POP) > EU(\neg POP)$.

4. SIMULATION RESULTS

To assess our user model, we ran text editing simulations with word prediction, as described in Section 3. The test text consisted of sentences drawn randomly from 10% of an unseen portion of the BNC. We sampled from a simulated user model based on the DBN described in Figure 2.

For each user type, we ran 100 simulations with texts about 200 words long. The averaged results show that the system's beliefs converged to the true type in all 36 cases. The time it took the system to reach convergence varied from about 20 to 150 words. Examples of convergence curves for three different user types (as a function of the number of observations) are shown in Figure 4.

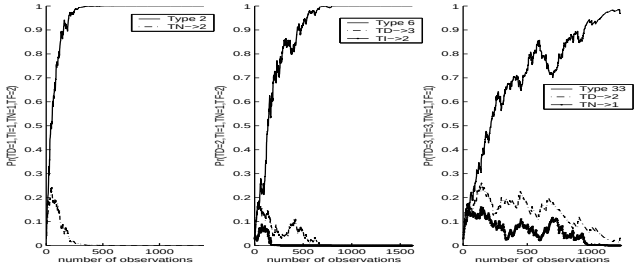

Figure 4: Examples of belief monitoring. Left: early convergence. Middle: convergence with respect to competing types. Right: slow convergence.

In our model, we chose an abstract representation of behavioural observations that is intuitive from a designer's perspective and that reduces the number of temporal dependencies (see the discussion in Section 3.1). Belief state monitoring is currently implemented in Matlab 6.5 R13. On average, this computation takes approximately 0.57 second on a Pentium M, 1.2G Hz CPU, 386 MB RAM processor.[3] When observational abstraction of the type used here is not feasible, or does not provide enough decomposition of the inference task to allow real-time belief state monitoring, approximation algorithms for monitoring (e.g., [4]) can be considered. Furthermore, belief update based on aggregate observations (e.g., every k steps) can also be used; since user state will generally evolve on much longer time scales than individual observational events, slight lags in user state estimation will generally have a negligible effect on performance.

A system's overall utility is quantified in terms of the rewards and costs it receives during its interaction with the user. In Section 3.3, we defined implicit reward and cost functions that vary according to the user's state in Section 2.3. Here, we use them to define the overall utility given the sampled user state and the actual quality of the suggestions, $U(F, N, TD, TI, QUAL)$:

$$U = \begin{cases} 0 & \neg\text{POP} \\ R(F,N,TD,TI,QUAL) & \text{POP and OBS = acc} \\ 0 & \text{POP and OBS = hh,hp} \\ C(F,N,TD,TI) & \text{POP and OBS o/w} \end{cases}$$

For each type, observed "reward patterns" reflect the system's adaptivity to the user's responses — more acceptances encourage more suggestions, and fewer acceptances fewer suggestions. Across user types, the patterns also show that more needy and dependent types receive higher overall utility, while more frustrated, distractible, and independent types receive lower utility. In Figure 5, we show some examples of the patterns of average accumulated rewards that the system receives for different types.

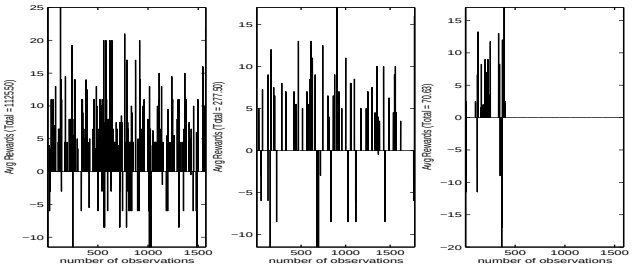

Figure 5: Examples of system behaviour according to inferred user type. Left: a user who welcomes help so the system offers them regularly. Middle: a sporadic user so help is sparse. Right: a user who rejects help so the system learns to back off.

For comparison purposes, we conducted experiments with other system policies. The policies we chose for this comparison are: suggest only if the quality is greater than a threshold (THRESH, for $QUAL > 3$), always make suggestions (ALWAYS), and never make suggestions (NEVER). We refer to our system policy as MEU. Table 2 compares average reward per time step for these different policies with respect to some representative user types. Generally, ALWAYS outperforms the other policies with dependent users who tend to need help, as we see in the first row of the table. However, it does poorly (often extremely) in all other cases. The second row shows that even with dependent users who are easily distracted or frustrated, the users may benefit more from the adaptive policies. In the remaining cases, an independent user, either easily frustrated or easily distracted or neither, benefits most from a system that learns to back off when help is undesired. These cases illustrate that a static policy, such as ALWAYS, or a policy that disregards the user type, such as THRESH, suffers most. Overall, MEU dominates THRESH for 17 of the 36 user types (sometimes quite significantly); for 12 of the types, MEU and THRESH perform comparably (within 0.05 of each other); and for 7 of the types, THRESH performs better than MEU, but only slightly. Although NEVER receives zero rewards in all the cases, it is unable to detect cases when the user in fact needs help, which is a state that could change from time to time.

5. LEARNING MODEL PARAMETERS

To replace the handcrafted parameters in the user model, we designed controlled experiments that explore different user states and logged corresponding user behaviour. Because user states are not directly accessible and cannot be explicitly elicited at every time step, our experiments collected data in a semi-supervised fashion.

[3]This prototype implementation can be considerably accelerated, so real-time inference is not a concern in this task.

Table 2: Comparison of policies using average rewards by user profile {TF,TN,TD,TI}

User Type	ALWAYS	MEU	THRESH	NEVER
{1,2,1,1}	1.64	0.93	0.91	0
{2,1,2,1}	0.46	0.62	0.65	0
{1,1,3,2}	-0.64	0.39	0.31	0
{1,1,1,3}	-10.29	-2.15	-2.29	0
{2,1,1,3}	-10.89	-1.89	-2.93	0
{2,1,3,3}	-6.04	-0.07	-1.43	0

5.1 Data Collection Experiments

Since most potential participants can type quickly without help, we designed a procedure that requires the user to type with a Dvorak keyboard. There were 45 users and each participated in 3 conditions. First, artificial delays of 2-5 seconds and sticky keys were introduced into the system at fixed intervals. The second condition presented a mix of audio and visual pop-up distractors at regular intervals. These distractors have a lifetime of 7 seconds and can end earlier if the user closes their residing windows. In the third condition, the text to be typed by the user contains a higher percentage of long words and esoteric vocabulary, as measured using the Fog index [13]. The first two conditions used text with Fog index = 11, while the third used Fog index = 30. To assess the user's current state, questions to elicit the user's current F and N values were posed at the end of each clause in all the trials. A post-questionnaire was designed to assess the user's general attitudes and tendencies under this computing environment so that we could elicit the user's type.

We developed a Java interface over the system described above. Based on informal observations, we identified a wide range of behaviours. For example, some participants ignored pop-up animations and audios, some laughed at them, while a few explicitly closed them. Participants also varied in their strategies for dealing with suggestions. Some typed one letter at a time while anticipating a suggestion and accepted it when it appeared, some buffered a few characters, typed them, and watched for suggestions, while others just did not accept the suggestions at all. Frustration was either not shown or appeared as a pause or sigh, but rarely as a physical action. We suspect this subtlety is influenced by the controlled environment and the presence of a researcher.

5.2 Parameter Estimation

The learning task at hand is known structure with incomplete data. With 45 participants and 3 sequences of observations each, there are $M = 135$ training cases. Each sequence on average consists of 845 observations, but ranges from 99 to 3097. Our goal was to learn the prior distributions F_0, N_0, $CONS_0$, the transition functions F_t, N_t, $CONS_t$, and the observation function, OBS_t. We applied a standard algorithm, expectation-maximization (EM) [8, 21]. The initial parameters were set randomly. EM iterates between computing the expected values of the hidden variables (given parameter estimates) in an E-step and maximizing the parameters given the data in an M-step, as follows:

- **E-step**: given $\hat{\theta}$ and data set $D = \{\mathbf{y}_l\}$, compute:

$$E_{Pr(\mathbf{x}|D,\hat{\theta})}(C_{ijk}) = \sum_{l=1}^{M} Pr(x_i^k, pa_i^j|\mathbf{y}_l, \hat{\theta})$$

- **M-step**: given the sufficient statistics, compute:

$$\theta_{ijk} = \frac{\alpha_{ijk} + E_{Pr(\mathbf{x}|D,\hat{\theta})}(C_{ijk})}{\sum_{k=1}^{r_i}(\alpha_{ijk} + E_{Pr(\mathbf{x}|D,\hat{\theta})}(C_{ijk}))}$$

where C_{ijk} is the number of times x_i^k and pa_i^j occur in the data set and α_{ijk} is a bias on the corresponding θ_{ijk}.

EM is guaranteed to converge to a local minimum. To avoid the sparse data problem and to incorporate prior knowledge, we used the handcrafted parameters in the simulations as biases in the M-step. We report on the training results using different weights in Section 5.3.2.

5.3 Model Comparison

In this section, we describe the procedures for discretizing the user variables and learning the parameters of the DBN.

5.3.1 Topology of Users

The post-questionnaire in our experiments elicited the values of TF, TD, and TI using 19 items, with each item intending to elicit a particular variable. Sample questions asked the user to self-report on a Likert scale whether they felt frustrated with the sticky keys, whether they were distracted by the pop-up animations, or whether imperfect suggestions were selected. Since this questionnaire was newly designed for this experiment, we carried out factor analysis on the responses to identify possible clusters and underlying factors [12]. Due to the small sample size (45), this analysis is a preliminary step in checking for strong correlations only. According to the Kaiser criterion, 4 factors had eigenvalue higher than 1.0; the scree test indicates 3 to 7 factors; using the percentage of variance explained, we obtain 3 factors for 74%, 4 factors for 84%, and 5 factors for 93%. Finally, we retain 3 factors by the interpretability criterion so that one factor corresponds to one design variable.

We used variance maximizing rotation to extract principal components. The resulting factor loadings confirmed that 10 items clustered with the intended factor, 2 items clustered incorrectly, and 7 items were undetermined. The incorrect items were reclassified into their clustered factors while the others maintained their original classes. Thereafter, we used the responses to compute the participants' score for TF, TD, and TI as $\sum_{i=1}^{l} \frac{r_i}{l}$, where r_i is a score and l is the number of items in the factor. By inspection, we partitioned the results into the domain of our model variables, i.e., 2 categories for TF and 3 for TD and TI.

For TN, we used typing speed (spd) as the motor attribute and the percentage of unfamiliar vocabulary (vocab) as the cognitive attribute in a user's neediness level. Let f_1 and f_2 be the normalized factor loadings for spd and $vocab$ respectively. Then $TN = f_1 * spd + f_2 * vocab$. By inspection, we partitioned the results into 2 categories for TN.

Based on this procedure, each participant has a user type profile {TF,TN,TD,TI} capturing their general tendencies in a computer setting. We plotted these profiles in Figure 6 to identify the types of users in our pool. As shown, our pool did not cover all the user types — the reason could be due to the small sample size or that some types do not exist. In many cases, we had one or two participants of a type (e.g., {1,2,2,1}). For type {2,2,3,3}, five participants had this profile. This plot suggests that users who are highly independent (TI=3) tend to get frustrated (TF=2) by the system. It also suggests a correlation between dependent and needy users (with TN negatively correlated to TI).

5.3.2 Parameter Settings

In training our model, we tried different weightings of the data and biases. The weights we used are: 0% of priors (i.e., data only),

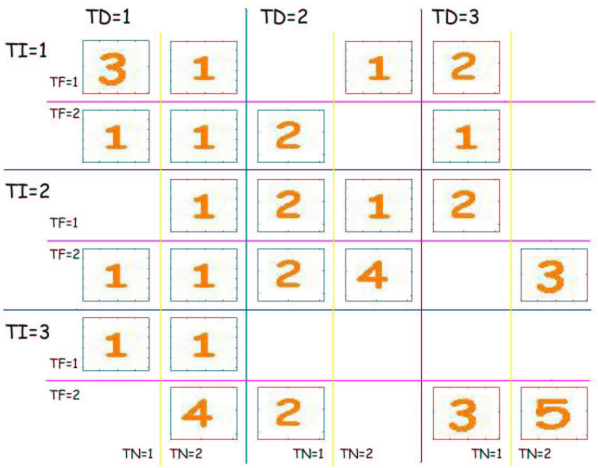

Figure 6: The topology of our participants. Each box indicates the number of users in that type.

10%, 30%, and 50%. If given enough representative data, we could compare these results using cross validation. However, our data set is very small relative to the state space so we discuss our choices informally.

With respect to the distributions using handcrafted parameters, we computed the Kullack-Leibler (KL) divergence [20] to assess the relative entropy of the learned distributions using $KL(P||Q_w) = \sum_{x \in X} P(x) log \frac{P(x)}{Q_w(x)}$, where P is the handcrafted distribution and Q_w is the learned one trained with weight w. The maximum KL divergence shows little difference among them: 3.1455 for the prior for CONS, 0.015 for the prior for N, and less than 0.01 for the others. If we had significant differences we could compare their performance further. However, with our results, we chose to use the learned distributions trained with $w = 10\%$.

6. USABILITY EXPERIMENTS

We designed a usability experiment to validate the learned model with real users by comparing their preferences with other system policies.

6.1 Pilot Study

We adopted a similar computing environment to the one used in data collection (cf. Section 5.1). In addition, the inference engine necessary to maintain the user model was implemented in Matlab. A Matlab-Java server was implemented so that the Java interface connects to it as a client.

The user's task was to copy 10 sentences into our editor using a Dvorak keyboard. These sentences were taken from an unseen test set (Fog index = 15). There are four conditions in this experiment, each employing a different system policy. The four policies we chose for this initial comparison are those used in the simulations (Section 4): THRESH, MEU (our system), ALWAYS, and NEVER. Participants were asked to type as accurately as possible. A questionnaire was given at the end of each condition and at the end of the entire experiment. In total, we had 4 participants.

6.2 Results

Since the users in the pilot are all novices with Dvorak (typing speeds between 4-8 wpm), they all preferred having as much help as possible. Contrary to other findings [9], one user commented that the system should provide completions even for short words like "and" and "to". We suspect that if we had users with a wider variation in typing speeds (i.e., different levels of neediness, TN), the results would reveal greater differences in their preferences.

Three users preferred ALWAYS to both adaptive strategies, which were in turn preferred to NEVER (i.e., "the more help the better"). This pattern is supported by the average percentage of characters typed using the four policies, as well as the subjective responses to whether a particular policy helped reduced effort and time. The actual time, however, revealed that typing with NEVER was the fastest, followed by the two adaptive strategies, and ALWAYS was the slowest. The fourth user preferred the two static strategies equally over the adaptive ones, because he could not predict exactly when the suggestions would appear (we discuss this further below).

Between the two adaptive strategies, we found that the overhead associated with the Matlab-Java engine[4] caused MEU to be ranked lower for two users. They added that if the system were faster, they would have preferred it over THRESH. All four users noted the quality of the suggestions in MEU was notably better than those in THRESH, although in three of the four cases, the percentage of correct suggestions were higher in THRESH. This suggests that the users are perceiving the utility of character savings as part of the quality of a suggestion. Also, the percentage of acceptances were higher in THRESH. This behaviour indicates they are dependent users in our model. Indeed, from plotting the system's inferred belief states, all four users were inferred as dependent ($TI = 1$) and needy ($TN = 2$). The number of characters typed using MEU was on average lower than those using THRESH.

Finally, about 20% of all the acceptances were partial suggestions, where users accept non-exact words and erase the endings. This indicates that the utility metric used in our language model (cf. Section 3.2) is more helpful than one that only uses a bigram statistic.

Despite the overall ranking by the users, which suggests that the adaptive strategies were not as useful as the static strategies, overall we find the results quite encouraging. Apart from a time-delay artifact, and concerns about predictability, user comments suggest that utility-based help is more desirable than suggestions made purely based on probability of acceptance. The task itself was also perhaps more difficult than anticipated, leading to a bias for ALWAYS, which may not be present over time (as skill levels improve) or in less unfamiliar tasks.

7. DISCUSSION AND CONCLUSIONS

We have outlined a general methodology for incorporating user models in automated assistance that encompasses a wide range of user types. In particular, we proposed to model user features explicitly so that they can be inferred and learned throughout over the course of interaction with the system. We demonstrated our approach in the word prediction domain via simulations with a handcrafted model and usability experiments after learning the parameters from extensive studies. Our results show that the model is able to adapt to different (static) user types and to evolving (transient) user state — changes in user attitude — during the course of the interaction. Although our model employs a myopic policy, its adaptive nature allows greater reward to be obtained over a wider range of user types than other fixed policies.

Of course, several drawbacks must be addressed within the general framework. One of the participants in the usability study in-

[4]Unfortunately, an artifact of the implementation created extra threading and file I/O delays; there is no inherent problem in the model itself that causes these delays.

dicated a preference for much more predictable system behaviour (a common theme in interface design). We can model these preferences as user features in the state space to reflect different attitudes toward usability goals. In this way, the cost model can weigh the amount of disturbance each action imposes on, say, a user's mental model of the application.

In an effort to exploit the sequential nature of human-computer interaction, we are currently exploring the construction of decision policies using *partially observable Markov decision processes (POMDPs)*. This model enables the system to evaluate the long-term impact of system actions optimally, giving the system the ability to take exploratory actions (for example) to directly learn about user types. Scalability is typically a concern for POMDP models, but recently they have come to be used in more and more realistic applications (see, e.g., work on using POMDP models in a prompting system for Alzheimer's patients that adapts to inferred user characteristics [2]).

Informal observations of the participants in our experiments clearly indicated a range of user types and the need to customize system response to account for their preferences. Feedback from several participants pointed to users having different reward (cost) functions, varying in value and structure. In particular, the usability results show that our adaptive policy was unable to always provide suggestions, even after learning that the user type is needy and dependent. This problem is caused by having predefined reward and cost functions that are insensitive to finer, numeric differences among individuals. To truly assess the system's overall utility, the user's reward function needs to be assessed or learned in real time. In the current system, system utility is a plausible function reflecting dependence on coarse-grained user types and a fixed model of character savings. Next steps in extending our methodology include incorporating means for more direct reward, cost, and utility model assessment tailored to individual users. Possibilities include using a distribution over a richer set of utility models, and updating this in response to observed user behaviour.

8. ACKNOWLEDGMENTS

We would like to thank all the participants in our experiments. This project was funded by NSERC, OGS, and PRECARN/IRIS.

9. REFERENCES

[1] D. Albrecht, I. Zukerman, and A. Nicholson. Pre-sending documents on the WWW: A comparative study. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 1274–1279, Stockholm, Sweden, 1999.

[2] J. Boger, P. Poupart, J. Hoey, C. Boutilier, G. Fernie, and A. Mihailidis. A decision-theoretic approach to task assistance for persons with dementia. In *Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence*, pages 1293–1299, Edinburgh, Scotland, 2005.

[3] T. Bohnenberger and A. Jameson. When policies are better than plans: Decision-theoretic planning of recommendation sequences. In *Proceedings of the International Conference on Intelligent User Interfaces*, pages 21–24, Santa Fe, NM, 2001.

[4] X. Boyen and D. Koller. Tractable inference for complex stochastic processes. In *Proceedings of the 14th Conference on Uncertainty in Artificial Intelligence*, pages 33–42, Madison, WI, 1998.

[5] A. Bunt, C. Conati, and J. McGrenere. What role can adaptive support play in an adaptable system? In *Proceedings of the International Conference on Intelligent User Interfaces*, pages 117–124, Madeira, Portugal, 2004.

[6] C. Conati, A. Gertner, and K. VanLehn. Using Bayesian networks to manage uncertainty in student modeling. *User Modeling and User-Adaptive Interaction*, 12(4):371–417, 2002.

[7] T. Dean and K. Kanazawa. A model for reasoning about persistence and causation. *Computational Intelligence*, 5(3):142–150, 1989.

[8] A. Dempster, N. Laird, and D. Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society, Series B*, 1(39):1–38, 1977.

[9] G. Foster, P. Langlais, and G. Lapalme. TransType: Text prediction for translators. In *Proceedings of the Association of Computational Linguistics Demonstrations*, pages 93–94, Philadelphia, 2002.

[10] K. Gajos and D.S. Weld. SUPPLE: Automatically generating user interfaces. In *Proceedings of the International Conference on Intelligent User Interfaces*, pages 93–100, Madeira, Portugal, 2004.

[11] P. Gorniak and D. Poole. Building a stochastic dynamic model of application use. In *Proceedings of the 16th Conference on Uncertainty in Artificial Intelligence*, pages 230–237, Stanford, CA, 2000.

[12] R.L. Gorsuch. *Factor Analysis*. Hillsdale, NJ: Lawrence Erlbaum, 1983.

[13] R. Gunning. *The techniques of Clear Writing*. New York: McGraw-Hill, 1968.

[14] T. Hasselbring and C. Glaser. Use of computer technology to help students with special needs. *The Future of Children: Children and Computer Technology*, 10(2):102–122, 2000.

[15] E. Horvitz and J. Apacible. Learning and reasoning about interruption. In *International Conference on Multimodal Interfaces*, pages 20–27, Vancouver, BC, 2003.

[16] E. Horvitz, J. Breese, D. Heckerman, D. Hovel, and K. Rommelse. The Lumière Project: Bayesian User Modeling for Inferring the Goals and Needs of Software Users. In *Proceedings of the 14th Conference on Uncertainty in Artificial Intelligence*, pages 256–265, Madison, WI, 1998.

[17] C. Huang and A. Darwiche. Inference in Belief Networks: A Procedural Guide. *International Journal of Approximate Reasoning*, 15(3):225–263, 1996.

[18] B. Hui, S. Liaskos, and J. Mylopoulos. Requirements Analysis for Customizable Software: A Goals-Skills- Preferences Framework. In *Proceedings of the 11th IEEE International Conference on Requirements Engineering*, pages 117–126, Monterey Bay, CA, 2003.

[19] A. Jameson, B. Großmann-Hutter, L. March, R. Rummer, T. Bohnenberger, and F. Wittig. When actions have consequences: Empirically based decision making for intelligent user interfaces. *Knowlege Based Systems*, 14(1–2):75–92, 2001.

[20] S. Kullback and R. Leibler. On information and sufficiency. *Annals of Mathematical Statistics*, 22(1):79–86, 1951.

[21] K. Murphy. *Dynamic Bayesian Networks: Representation, Inference and Learning*. PhD thesis, Department of Computer Science, UC Berkeley, CA, USA, 2002.

[22] F. Shein, T. Nantais, R. Nishiyama, C. Tam, and P. Marshall. Word cueing for persons with writing difficulties: WordQ. In *Proceedings of the 16th Annual International Conference on Technology and Persons with Disabilities*, Los Angeles, 2001.

[23] X. Zhou and C. Conati. Inferring user goals from personality and behaviour in a causal model of user affect. In *Proceedings of the International Conference on Intelligent User Interfaces*, pages 211–218, Miami, 2003.