Adapting Interaction Environments to Diverse Users through Online Action Set Selection

M. M. Hassan Mahmud
School Of Informatics
University of Edinburgh, UK

Benjamin Rosman
Mobile Intelligent Autonomous Systems
CSIR, South Africa

Subramanian Ramamoorthy
School Of Informatics
University of Edinburgh, UK

Pushmeet Kohli
Machine Learning and Perception
Microsoft Research Cambridge, UK

Abstract

Interactive interfaces are a common feature of many systems ranging from field robotics to video games. In most applications, these interfaces must be used by a heterogeneous set of users, with substantial variety in effectiveness with the same interface when configured differently. We address the issue of personalizing such an interface, adapting parameters to present the user with an environment that is optimal with respect to their individual traits - enabling that particular user to achieve their personal optimum. We introduce a new class of problem in interface personalization where the task of the adaptive interface is to choose the subset of actions of the full interface to present to the user. In formalizing this problem, we model the user as a Markov decision process (MDP), wherein the transition dynamics within a task depends on the *type* (e.g., skill or dexterity) of the user, where the type parametrizes the MDP. The action set of the MDP is divided into disjoint set of actions, with different action-sets optimal for different type (transition dynamics). The task of the adaptive interface is then to choose the right action-set. Given this formalization, we present experiments with simulated and human users in a video game domain to show that (a) action set selection is an interesting class of problems (b) adaptively choosing the right action set improves performance over sticking to a fixed action set and (c) immediately applicable approaches such as bandits can be improved upon.

Introduction

Interactive interfaces, such as natural gesture-based user interfaces, have revolutionised the way we interact with video games, robots and other applications. A key attribute of such an interface is the way in which users' high-dimensional movements are interpreted and correspondingly mapped to the command set within the application. Many modern software applications have similar requirements. For instance, interfaces to search engines (e.g., Google, Bing), or photo-editing software (e.g., Adobe Photoshop), involve numerous configuration choices that, implicitly or explicitly, define the context of the interaction between the human user and the computational process actually being performed.

Almost all of these applications must be deployed with large user populations, with substantial diversity in the performance that results from any given pair of user and configuration setting. For instance, even with a simple interface like joystick-based navigation within a video game or field robotics application, there is variety in dexterity and skill. A mismatch between a user's skill level and settings such as the sensitivity level at which one interprets a particular motion as a signal can have substantial negative impact on the user's ability to perform a task. Sometimes the mismatches can be in assumptions about perceptual ability, e.g., how precisely aware a user is regarding the current computational context. As behavioural studies, e.g., Valtazanos and Ramamoorthy (2013), have shown, perceptual limitations can have a disproportionately large effect on user's performance when the task involves interactive decisions in a dynamic environment.

These issues of personalisation in diverse applications are unified in the computational problem of automatically shaping the environment so that it is best adapted to the specific user whom the system is currently facing. Our particular interest is in adapting online, to personalise on the fly as the user interacts with the system, without long calibration phases and with only partial knowledge of the underlying traits that induce variability.

The objective of this paper is to concretely define this problem of user environment shaping by presenting a model of this problem. We also present experimental studies in a stylised video game domain with human and simulated users to characterise the benefits of adaptation as compared to using some a-priori optimal action set. We posit some desiderata such adaptation algorithms should satisfy: it must afford - *safety*, i.e., ensuring that individualised adaptation does not lead to worse performance than a pre-configured statically optimal setting, *efficiency*, i.e., learning to obtain settings that enable the user achieve their maximal performance, and *speed*, i.e., the ability to adapt within a few episodes which roughly corresponds to users' patience. We also present a simple algorithm that satisfy these desidirata, and show that more established bandit algorithms such as Auer (2002) do not. This leaves opens the door for development of more powerful algorithms in future work.

We now briefly present our approach to modelling the problem of interface adaptation by environment shaping. We model the user as a sequential decision maker, operating in an Markov decision process (MDP) (see Sutton and Barto (1998), Puterman (1994)), where the MDP models the task.

We idenitify an interface for the task with a *subset* of the set of MDP actions. The adaptation now takes the form of choosing the *action subset* the user is allowed to use when accomplishing the task. We posit that different user *types* (e.g. skilled, expert, risk averse, daredevil etc.) will accomplish the task in different ways, which will manifest as different transition distributions of the MDP. So for different user type/transition distribution different action sets might be optimal. The task of a learning algorithm in this problem will be to combine estimation of observed transition distribution with prior training with users, and then determine online which interface/action-set is the optimal and present that action-set to the user.

Related work

The problem of learning to adapt to user behaviour is garnering increasing attention from researchers in different domains.

HCI researchers have considered ways to customize user interfaces. For instance, Gajos, Wobbrock, and Weld (2008) present a method for automatically synthesizing user interfaces tuned to user (dis)abilities. The process involves offline calibration using a battery of tasks, based on which one obtains information necessary to pose the interface design problem as one of combinatorial search. Related works on personalisation, e.g., in this sense, our work is related to recent work on personalisation. Seuken et al. (2012); Zhang, Chen, and Parkes (2009) have also considered ways to model user performance in decision and game theoretic models, to enable optimization of parameters therein. We take inspiration from these notions of identifying a user interface with parameters describing user performance but our primary focus is on continual online adaptation.

A different line of work that is closely related to our approach is on plan recognition, e.g., see Charniak and Goldman (1991), where one utilizes traces of an agent's actions in order to infer which one of many possible plans they are attempting to execute. Recent instantiations of this type of problem have included work such as Fern and Tadepalli (2010) on hidden-goal MDPs. Our focus is different in that we are not trying to predict the user's future state, based on the past actions; instead, we try to infer latent 'personality traits' or skill/preferences of the user (as in the earlier referenced HCI work) in order to shape their environment – to enable them to do better in their chosen task.

Some other related work in this area includes Hoey et al. (2010), who study the problem of synthesizing a policy for issuing prompts to dementia patients that is best adapted to an unobserved 'attitude' that must be inferred. In that work, similarly to Fern and Tadepalli (2010), the goal is to influence an agent with latent states to move towards a mutually desired goal state and efficiency is determined by optimally coordinating with such a user. More directly related to our problem of user interface adaptation, Hauser et al. (2009) present the concept of website morphing, where the configuration of a web interface are adapted in response to estimated 'cognitive state' of a user. All of these works inspire our own, but our model differs from these prior works in that our adaptation is at the level of changing the *action set* as a whole, as opposed to intervening at individual states to influence the path taken within the 'inner loop' of the task. It is perhaps also noteworthy that the algorithm we adopt in our experiments, which poses the problem as one of online model selection, is relatively easy to implement and is attractive from a computational cost standpoint. That said, the algorithm as present in this paper is merely a first step and many more computational procedures can be defined, as in these prior works, to implement our model of action set selection. This includes consideration, e.g., of users' learning behaviour (Rosman et al. (2014)) and other forms of changing dynamics.

Interface Adaptation As Action Set Selection

In our problem, the user is an *agent* acting according to a Markov decision process (MDP) (see Puterman (1994)) and invokes optimal policy [1]. The user skill level/type determines the transition function of this MDP – this models the fact that, for instance, a less skilled user will have a different way of transitioning between states (e.g., being more variable) than an expert. Unlike in standard applications of optimizing policies for given MDPs, we will be changing the action set α as the interaction proceeds in episodes, which is the main point of the interaction shaping process. The goal of the learner [2] will be to choose action sets so as to maximize the future expected discounted reward of the agent. So, the search space for the main learning algorithm in this paper is a type-space[3], rather than the space of paths the user actually traverses, and the value of a type is the value of the optimal policy given the type. In the following we first describe MDPs and then we describe our model.

Markov Decision Processes

Markov decision processes are a popular model for capturing sequential decision problems – for an introduction to Markov decision processes, see Puterman (1994) Sutton and Barto (1998). A finite MDP $\mathcal{M}$ is defined by the tuple $(\mathcal{S}, \mathcal{A}, \mathcal{R}, T, R, \gamma)$ where $\mathcal{S}$ is a finite set of states, $\mathcal{A}$ is a finite set of actions and $\mathcal{R} \subset \mathbb{R}$ is the set of rewards. $T(s'|s,a)$ is a the state transition distribution for $s, s' \in \mathcal{S}$ and $a \in \mathcal{A}$ while and $R(s,a)$, the reward function, is a random variable taking values in $\mathcal{R}$. Finally, $\gamma \in [0,1)$ is the discount rate. A (stationary) policy π for $\mathcal{M}$ is a map $\pi : \mathcal{S} \to \mathcal{A}$.

The Q function $Q^\pi : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ of a policy π is defined by: $Q^\pi(s,a) = \mathbb{E}[R(s,a)] + \gamma \sum_{s'} T(s'|s,a) Q^\pi(s', \pi(s'))$ and gives the total future expected discounted reward obtained by taking action a at the current step, and then following the policy π after that. The value function for π is defined as $V^\pi(s) = Q^\pi(s, \pi(s))$ and is total future expected discounted reward obtained by following policy π. An optimal policy π^* is defined as $\pi^* = \arg\max_\pi V^\pi$ – the Q function is given by $Q^*(s,a) = \mathbb{E}[R(s,a)] + \gamma \sum_{s'} T(s'|s,a) \max_a Q^*(s',a)$. For the optimal policy the value functions is denoted by V^*. In general reinforcement learning, goal of the agent is to estimate Q^* and then choose the action $\arg\max_a Q^*(s,a)$ at state s. In the sequel, we assume, without loss of generality, a fixed starting state s° for

a MDP and define $V^\pi \triangleq V^\pi(s^\circ)$.

Our Model of Interface Adaptation

From our discussion in the Introduction, our model of the interface adaptation for applications we are interested in should allow us to: (i) represent each user as a sequential decision maker trying to optimize some goal; (ii) express that users can be of different types; and (iii) specify interfaces as collection of actions that the user is allowed to choose from at any given point in time. In this section we give a concrete definition of a model that fulfils these criteria.

It seems like a plausible way to satisfy criteria (i) is to model the user as a MDP as it is a model of sequential decision making where the user has a very precise objective to optimize. In this case, the states $\mathcal{S}$ of the MDP represent the different stages of a task, the actions $\mathcal{A}$ represent the choices available from all the interface, and the reward R gives feedback on how to what extent the user has accomplished task. The transition T is used to satisfy criteria (ii).

In particular, given the above mapping from MDPs to users+tasks, we can make our model satisfy criteria (ii) by positing a *class* of MDPs parametrized by the user type. We assume that the user of our interface has a type τ from a finite set of types $\boldsymbol{\tau}$, and the transition function T has the form $T(s'|s,a;\tau)$. This models the fact that depending on the user type, the same action will have different effect on which state of the task is reached. For instance, in a joystick interface for teleoperating a robotic unit, a 'forward' action with the joystick set to high sensitivity will have different reactions depending on the speed of reflex (i.e. type) of the user. So, our user+task space is modelled by a class of MDPs M of the form:

$$\mathsf{M} \triangleq \{((\mathcal{S}, \mathcal{A}, \mathcal{R}, T(\cdot;\tau), R, \gamma) | \tau \in \boldsymbol{\tau}\} \tag{1}$$

We now discuss how to incorporate (iii), the availability of interfaces, into this picture.

To satisfy criteria (iii) we recall that each interface corresponds to a set of actions that are available to the user. In that case, it makes sense to posit that the action set $\mathcal{A}$ has been divided into a subsets $\alpha^1, \alpha^2, \alpha^n$, where each α^i corresponds to an interface and contains the actions available in that interface. So for instance, if the interface is a website, then each α^i corresponds to a specific view of the website, and α^i contains the buttons, text fields, instructions etc. available at that view. If the interface is a video game interface or the joystick for a teleoperated robot, then each α^i may contain motion actions available at a specific level of sensitivity. Putting this all together, this means that the action set $\mathcal{A}$ is a union of subsets $\alpha^1, \alpha^2, \cdots \alpha^n$ where in general we expect each α^i to be significantly smaller than $\mathcal{A}$ (we do not assume that the action sets α^i are disjoint).

To model the fact that a user only gets to use one interface at a time, we require that only one set α^i is *live* at any point in time, so that the user is only allowed to choose actions from the live set. Given a live set α_t at time step t, we assume that the user chooses the action $\pi^*_{\alpha_t}(s_t)$ where is s_t is the state at step t and $\pi^*_{\alpha_t}$ is the policy maximizing $V^\pi(s_t)$ among all policies which only chooses actions from α_t.

The Learning Problem

Given the above concrete formulation of our problem, we can define our learning problem as follows. The agent solves a sequence of MDPs from the set M in collaboration with the learner – that is the goal of both the agent and the learner is to maximize the future expected discounted reward of the agent. At the beginning of each phase of the problem, we begin with some action set α_0 live and a user with a particular type τ^* comes in to use the application. At each step t the user chooses an action a and in response the learner decides to make a (possibly different) interface α_{t+1} live for the next phase. The learner cannot observe the true τ^* but knows the value of T for each τ. Once α_t is made live, the agent learns and acts according to the policy $\pi^*_{\alpha_t}$.

We can define two fixed strategies for live action set selection which are used as reference strategies later. The first is the 'true' optimal action set and the second one is the *a-priori* optimal action sets:

$$\alpha^* \triangleq \arg\max_\alpha V^\alpha_{\tau^*}, \qquad \alpha_* \triangleq \arg\max_\alpha \mathbb{E}_{W(\tau)}[V^\alpha_\tau] \tag{2}$$

So α^* is the action set/interface that we would have chosen if we knew the true user type τ^* from the beginning. And α_* is the optimal action set based on some prior W over the set of user types.

Experiments

Our experiment is a stylised version of the problem arising in popular mobile device games such as Temple Run (ima), where the objective is to guide a continuously moving character across a terrain and through a series of obstacles by swiping on the screen, tilting the mobile device etc. A goal might be to induce the player to play as long as possible. This calls for adaptation, e.g., between the speed of the game-character and user skill in terms of reflexes on such devices. We frame this adaptation as a simple instance of live action-set selection where each possible speed of the game-character corresponds to an action set. The actions in a particular action set are the same (swipe on screen, tilt device) but the effect is different, depending on the action set. This is also an illustration of a situation where it is obviously better to have distinct action sets rather than having all the actions together simultaneously – having all the swipe and tilt actions at different speeds available at the same time would make the game extremely hard, and reduce its enjoyability significantly. It is worth repeating here that games such as Temple Run are the simplest instances of our problem. As we are focused more on introducing a particular class of interface adaptation problem, this suffices. Our hope is to in future address much more challenging and dynamic problems like the ones we presented in the introduction.

In the following, we first describe our domain, which distils the essential interface adaptation issues in the above

[2]We refer to learning algorithms, which choose the action sets as the *learner*; the human user is often referred to as the *agent*.

[3]The term type-space has a broader interpretation in the game theory literature than in our work; here a type is a categorization over policies, hence possible behaviours.

domain. We then describe the algorithms that we compare against and then we present our results.

The Red Ball Video Game Domain

The domain that we use is a simple video game where the user has to control a red ball from a target location to a goal location (see Figure 1). The ball speed can be set to one of several possible values and the player/user can only control the direction of motion. Her goal is to take the ball from a start location to the goal location as quickly as possible without bumping into walls. The user has several possible skill levels in terms of controlling the ball. So, depending on her skill, different speeds are appropriate.

Specifically, the ball moves with a continuous speed p pixels-per-second and taking an action in the each of the cardinal directions N, S, E, W changes the motion of the ball to that direction. The speed p is one of $\{30, 40, 50, 60, 70\}$. We assume that the users are different level of skills in terms of controlling the ball. Given that this game is simple, we artificially enforce this difference in skill level by giving each user a different ball sizes of b pixels where $b \in \{2, 5, 10, 20, 40, 60\}$. We also add a noise level of $b/2\%$ to the motion of the ball to further enforce different performance. Hence, we capture two types of limitations in user behaviour. The size of the ball reflects limits to perceptual ability, with large ball sizes indicating reduced acuity. The noise of the motion of the ball reflects limits to motor ability, such as intrinsic jitter in the way they use the device. Combinations of larger ball+noise imply less skill. In our experiments, neither of the algorithms EXP-3 and BOA (described below) know the true ball size during the testing phase.

MDP Formulation of Interaction Adaptation

To construct our MDP formulation, we need to specify state space $\mathcal{S}$, actions $\mathcal{A}$, rewards $\mathcal{R}$, transition function T, reward function R and the set of action sets AS. $\mathcal{S}$ is the location of the ball at each time step (the game field was 1000×1000). Each action set α_p corresponds to all the motions at speed p. The actions $a \in \alpha_p$ were the motions in each of the cardinal directions N, S, E, W at speed p.

The direction of motion at step t is the most recent direction chosen by the user at $t' \leq t$ (so, the actual play of the user is modelled in the MDP as the user choosing the same action at every step as her last choice of direction). There was a constant amount of noise (on top of noise due to skill/type) with each action, so the ball moves in the given direction followed by (discretized) Gaussian motion (perpendicular to the direction of motion) with mean 0, and $\sigma \in [0.3, 8]$ (chosen randomly for each episode).

The transition function T is defined as follows. There were 5 different types: ball size $b \in \{2, 5, 10, 20, 40, 60\}$, noise levels $n_b = b/2\%$. So for type $\tau = (b, n_b)$, with probability $1-n_b$, the ball moves in the current direction, and with probability n_b, it moves in another random direction. Hence, the type modifies the base transition probabilities (1) by the noise n_b and (2) because the ball sizes determine the locations that result in collisions. Finally for the reward function R, the player receives a reward of -1 for every time step and -5 for every collision.

Experiment Description

We performed two sets of experiments on the above domain. In the first set, the user is a simulated, while in the second set the users are humans. In both cases we ran a training phase and test phase. The data collected during the training phase was used to train our algorithms and these learned algorithms were used in the testing. During the training phase we provided transition distribution information for each $(b, n_b) \times z$ (see sections below for more detail). The simulated user policy was the A^* algorithm, where the direction chosen at each step, given speed p, was toward the location, at distance p from the current location, that had the lowest cost.

During the test phase we chose the type at random and recorded the loss (difference between the user's performance and optimality) over 5 episodes (plays of the game) for various algorithms and policies. Note that, the user knows her type but *our algorithms do not*. For the simulated user, the policy used was the same, and for the human experiment, we used three different users, each performing a sequence of many trials involving combinations of ball sizes and speeds.

The strategies we used were α^* (choose the optimal action set (2) for the true type), α_* (choose the a-priori/population optimal action set (2)), and the algorithms we used were the EXP-3 algorithm for bandits (Auer (2002)), and our own algorithm that chooses the *Bayes optimal action set with* additional exploration. We describe these last two in more detail now.

Algorithms

It is very natural to formulate the problem of live action set selection as a non-stochastic multi-armed bandit problem (NSMB). In the bandits problem, there are c *arms* where each arm i has a *payoff process* $x_i(t)$ associated with it. The learner runs for M steps and at each step t needs to *pull*/select one of the arms $f(t)$ and his payoff is $x_{f(t)}(t)$. Additionally, the learner only gets to view the payoff of the arm $f(t)$ it has chosen. The goal of the learner is to minimize its *regret* with respect to the best arm, that is minimize the quantity $\max_i \sum_{t=1}^{M} x_i(t) - \sum_{t=1}^{M} x_{f(t)}(t)$. One optimal algorithm in the general case for minimizing the *expected regret* is the EXP-3 algorithm mentioned above. For live action-set selection, we can identify each action set α as an arm and the payoff received for choosing the arm as the reward in the following time-step. With this identification we can EXP-3 exactly as described in Auer (2002).

The second algorithm we use is a simple algorithm, which we call BOA, that with probability ϵ makes live an action set at random, and with probability $1 - \epsilon$ makes live the *Bayes optimal action set* at step t (hence the name BOA for the algorithm) . This action set defined as follows. Assume that at time step t of a particular interaction session we have observed a state-action sequence $sa_{0:t} \triangleq s_0 a_0 s_1 a_1 \cdots s_t$ (with $sa_{0:0} \triangleq s_0$). Given this session, the likelihood of a type τ_i is $L(\tau_i | sa_{0:t}) = \prod_{i=0}^{t-1} T(s_{i+1} | s_i, a_i; \tau_i)$ (the function $T(|; \tau_i)$ is learnt during the training phase. The posterior probability of τ_i is $Pr(\tau_i | sa_{0:t}) = L(\tau_i | sa_{0:t}) W(\tau_i)$ where the prior $W(.)$ is the frequency with which we observed τ_i during

training. The Bayes optimal action-set α_{BO} is the one that maximizes the expectation of V_τ^α with respect to the posterior over $\boldsymbol{\tau}$:

$$\alpha_{BO}(sa_{0:t}) \triangleq \arg\max_\alpha \sum_i Pr(\tau_i|sa_{0:t})V_{\tau_i}^\alpha \quad (3)$$

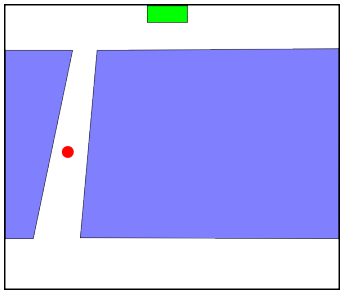

Figure 1: The Red Ball video game domain; green rectangle is the goal location, and the red circle is the ball. The initial location was a random location in the bottom.

Results

Data Used For Experiments. For the simulated user case, during training we generated 1548 trajectories from which we learned the $T(\cdot|\cdot;(b,n_b))$ for each type of ball-size+noise (b,n_b). The trajectories were divided uniformly over the different types. The results we present in graphs below are for the testing phase, which contained 344 trajectories with the types draw uniformly at random before each trajectory. For the human experiments we used three different human users. For the training phase we used 311 different trajectories and for the testing phase we used 51 trajectories and the results below are based on these latter ones.

Results For Simulated User. The results are presented in Figures 2 – 4. Figure 2 shows the loss of each p for three representative (b,n_b) and establishes the baseline performance of action-sets for these types: higher speeds tend to work better for smaller sizes and conversely. The baseline performance corresponds to the user choosing the optimal policy given the speed and her type

Figure 3 shows that adaptation is more beneficial when we have greater number of types of users in our pool of users. Here, in the x-axis each x corresponds to all possible combinations of x types (so all $\binom{6}{x}$ combinations). This captures different levels of heterogeneity in user populations. In the figure, for the curve α_*, the point $\alpha_*(x)$ gives the loss of best speed averaged across all populations of size x. Hence $EXP\text{-}3(x)$ gives the loss of EXP-3 averaged over all population of size x, while $BOA(x)$ gives the loss of BOA averaged over all population of size x. This curve shows, that averaged across population sizes, BOA outperforms EXP-3 significantly. Furthermore, while for the smaller population size, the α_* beats BOA, for the larger population sizes BOA beats α_* comprehensively, illustrating the need for adaptation. Additionally, note that the curve for α^*, the optimal per type, averaged over all types, is the line $f(x) = \alpha_*(1)$ because $\alpha_*(1)$ gives the 'population optimal' of size 1, which is just α^*.

Finally, Figure 4 shows BOA is able to identify types very quickly. In particular, the y-axis gives the ratio $\frac{L(\tau^*|sa_{0:t})}{L(\hat{\tau}_t|sa_{0:t})}$. Here $\hat{\tau}_t \triangleq \arg\max_\tau Pr(\tau|sa_{0:t})$, i.e. the maximum a-posteriori type. t is the time step at which the ratio is being computed (this is measured by episodes in the x-axis for convenience, where each episode lasts for M steps). Hence, since the likelihood $L(\tau|sa_{0:t})$ measures how well the data supports that τ is a true type, Figure 4 shows that BOA generally converges to the true type rapidly. However, it also shows that BOA has trouble identifying the smallest type. We conjecture that this is because the behaviour of $b = 5$ and $b = 2$ are nearly indistinguishable. In other cases, correct type identification, occurring almost always halfway through the first episode.

Results for Human Users. The results for the human experiments are presented in Figures 5 – 7 and they draw similar conclusions as with the simulated user. Figure 5 establishes the baseline as before and shows different speeds are optimal for different types.

Figure 6 shows the performance of BOA and α_* for the human experiments in terms of population diversity. We do not report the performance of EXP-3 because data collection for that algorithm takes time that exceeded the constraints of our human subjects. In these experiments, we see that BOA significantly outperforms α_* for all population sizes, hence demonstrating the benefit of adaptation with real human subjects, corroborating our more extensive simulation results above. This is a key experimental result of this paper.

Finally, Figure 7 shows the efficacy of our algorithm at identifying different types. As in simulated user experiments, our algorithm has some trouble distinguishing between the very nearby types $b = 5$ and $b = 2$ but the error is not beyond the threshold of small noise.

Interpreting the Experiments: Safety, Efficiency and Speed

As stated in the introduction, our goal is to construct algorithms satisfying these desiderata. Our experiments show that the algorithm BOA is safe. For simulated users, BOA does not under-perform w.r.t. any alternatives including EXP-3 and static best speed α_*. In human experiments, BOA outperforms α_* (Figure 6). In artificially constrained low-population-diversity setting (1-3) with simulated users, α_* is marginally better than BOA; but BOA outperforms at realistic levels of diversity (Figure 3). Hence, a key conclusion of our experiments is that *this kind of adaptation is indeed necessary in environments with substantial diversity as the regret of alternate approaches shoots up at high diversity*. We also notice that the results of EXP-3 imply that there is substantial scope for development of new algorithms in this application. BOA is also efficient, in particular, Figures 3 and 6 shows that our solution adapts differently for different user types. Finally, Figures 4 and 7 also shows that BOA has speed, in that we identify the true types fairly rapidly, demonstrating speed.

SIMULATED USERS

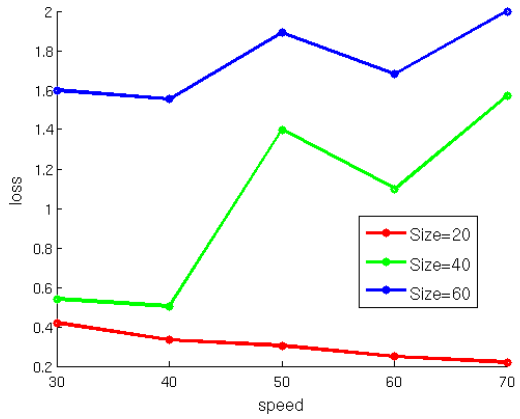

Figure 2: The performance of each speed for each type establishing baseline domain performance.

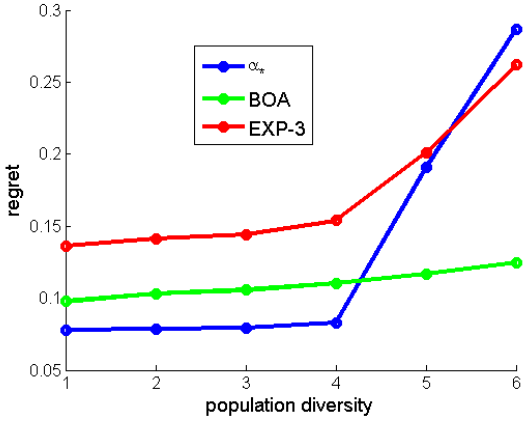

Figure 3: The performance of best static speed α_*, EXP-3 and BOA for each of 6 possible combinations of type populations when the user is simulated.

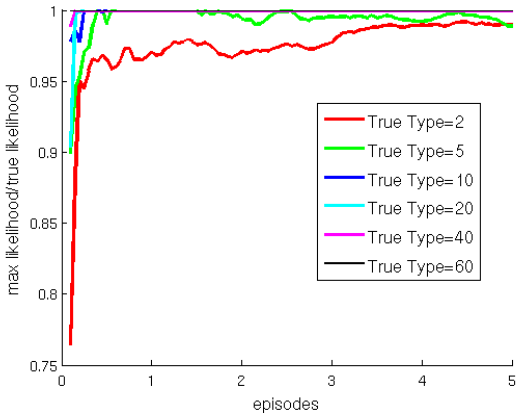

Figure 4: Rapid convergence to the true type, shown by the ratio of the likelihoods of the maximum-a-posteriori type and the true type, averaged over all population-diversity trials in our experiments when the user is simulated. The likelihoods are sampled uniformly at 100 points during each episode (game).

HUMAN USERS

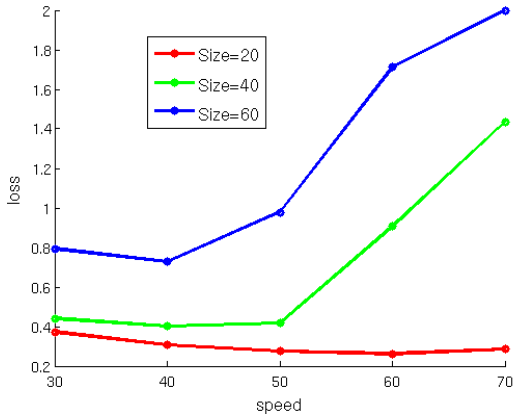

Figure 5: The performance of each speed for each type, establishing baseline performance in domain.

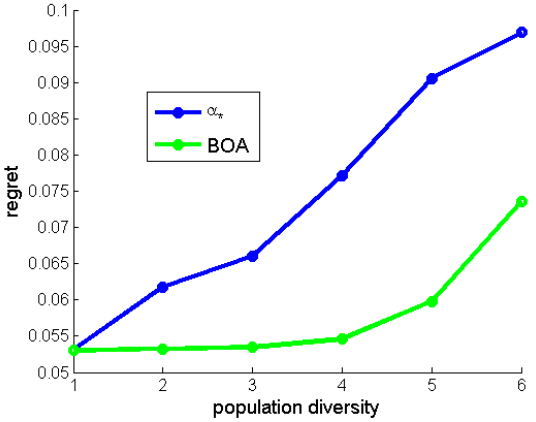

Figure 6: The performance of best static speed α_* and BOA for each of 6 possible combinations of population types.

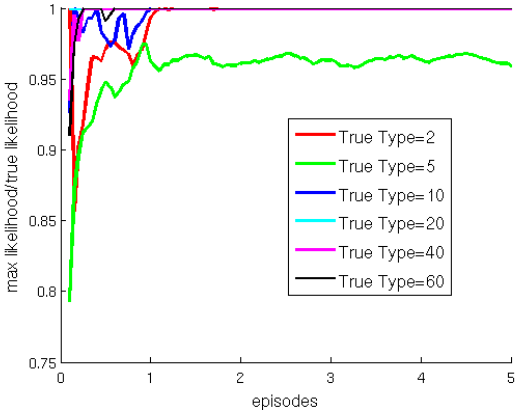

Figure 7: Rapid convergence to the true type, shown by the ratio of the likelihoods of the maximum-a-posteriori type and the true type, averaged over all population-diversity trials in our experiments when the user is human. The likelihoods are sampled uniformly at 100 points during each episode (game).

Conclusions

The main contribution of this paper is a new model for interface adaptation, based on categorising user types, each corresponding to a certain decision process that describes the user's skill, and then posing the problem of adapting to an unknown user as one of Bayesian model selection in an online setting. We perform model selection in the setting where the user changes her behaviour immediately when the action set changes. While this assumption has not been substantially contradicted in our experiment, it is likely that deviations may be more substantial as the complexity of the task goes up. A direction for future work is to incorporate models of user learning, e.g., Rosman et al. (2014), into this model selection process. Similarly, we work in a setting where a training corpus corresponding to various types seeds our learning algorithm. We envision a number of different ways of passively acquiring such training corpora from users' regular behaviour with test systems. However, here again, incrementally acquiring data corresponding to heterogeneous types and performing (fully unsupervised) learning in a lifelong fashion is an important direction to be explored in future work.

Acknowledgements

This work has taken place in the Robust Autonomy and Decisions group within the School of Informatics. Research of the RAD Group is supported by the the European Commission through SmartSociety Grant agreement no. 600854, under the programme FOCAS ICT-2011.9.10.

References

Auer, P. 2002. Using upper confidence bounds for exploitation-exploration tradeoffs. *Journal of Machine Learning Research* 3:397–422.

Charniak, E., and Goldman, R. 1991. A probabilistic model of plan recognition. In *Proceedings of the ninth National conference on Artificial intelligence - Volume 1*, AAAI'91.

Fern, A., and Tadepalli, P. 2010. A computational decision theory for interactive assistants, advances in neural information processing systems. In *Proceedings of the* 23^{rd} *Conference on Neural Information Processing Systems*.

Gajos, K.; Wobbrock, J.; and Weld, D. 2008. Improving the performance of motor-impaired users with automatically-generated, ability-based interfaces. In *CHI '08: Proceeding of the twenty-sixth annual SIGCHI conference on Human factors in computing systems*, 1257–1266.

Hauser, J. R.; Urban, G. L.; Liberali, G.; and Braun, M. 2009. Website morphing. *Marketing Science* 28(2):202–223.

Hoey, J.; Poupart, P.; von Bertoldi, A.; Craig, T.; Boutilier, C.; and Mihailidis, A. 2010. Automated handwashing assistance for persons with dementia using video and a partially observable markov decision process. *Computer Vision and Image Understanding* 114(5):503–519.

Temple run. *Developed and Published by Imangi Studios, 2011*.

Puterman, M. L. 1994. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley and Sons.

Rosman, B.; Ramamoorthy, S.; Mahmud, M. H.; and Kohli, P. 2014. On user behaviour adaptation under interface change. In *Proceedings of the International Conference on Intelligent User Interfaces (IUI)*.

Seuken, S.; Parkes, D. C.; Horvitz, E.; Jain, K.; Czerwinski, M.; and Tan, D. S. 2012. Market user interface design. In *ACM Conference on Electronic Commerce*, 898–915.

Sutton, R. S., and Barto, A. G. 1998. *Reinforcement Learning: An Introduction*. Cambridge, MA: MIT Press.

Valtazanos, A., and Ramamoorthy, S. 2013. Evaluating the effects of limited perception on interactive decisions in mixed robotic environments. In *HRI '13: Proc. ACM/IEEE International Conference on Human-Robot Interaction*.

Zhang, H.; Chen, Y.; and Parkes, D. C. 2009. A general approach to environment design with one agent. In *IJCAI*, 2002–2014.