Interactive Preferences and Decision-Theoretic Planning

Derek Williams*, Kyle Bailey†, Alex Dekhtyar*, Judy Goldsmith*, Beth Goldstein‡, Raphael Finkel*, Joan Mazur§

Abstract

We introduce a suite of interlinked software tools for eliciting preferences, doing decision-theoretic planning, and displaying the plans. The software allows a user to walk through possible trajectories, adjust preferences, and compare potential trajectories. This paper focuses on the elicitation process, the plan display, and interactions between the two.

1 Introduction

> **Plan** always involves a mental formulation and sometimes graphic representation: It often suggests a particular pattern and some degree of achievement or harmony. [6]

Decision-theoretic planning software for a wide range of applications (from island evacuation [3] to academic advising to medical planning [2; 9; 11]) should react to changes in the planning environment. These changes can occur both in the external world and within a user. Changes within a user are often prompted by observing the plan that has been submitted for review. Thus, planning should be a highly interactive process — user preferences form the utility function for the planning software; the output plan is displayed to the user; upon observing the plan, the user modifies their preferences and repeats the process.

In our interactive planning system, information flows from user to preference elicitation tool to the planner to the plan-display tool, then back to the user, forming a feedback loop that repeats during plan-building until the user is satisfied with the proposed plan. In this paper, we focus on the first and the last software components of this system: the preference elicitation tool (POET) and the plan display tool (PlanScan) and on the direct and indirect interaction between them.

The feedback process using these tools proceeds as follows. Together with the plan, PlanScan receives from the planner the information about the user preferences used in planning. During plan examination, the user may alter preferences using POET and request a new plan from PlanScan, which can display different plans and their associated preferences in different tabbed panels. The user can view and compare plans for different preferences.

PlanScan allows the user to specify a particular state and start the plan walkthrough from that state. This feature aids in the comparison process. It also allows the user to choose an action *not recommended by the plan* for comparison purposes.

In this paper, "plans" are long-term plans or policies for action in a stochastic domain. The domain is assumed to be modeled as a factored Markov decision process, where a state is defined as a setting of values for all of the fluents (also known as state variables or attributes). Policies are mappings from states to a finite set of actions.

What makes such plans difficult for a user to comprehend is that each action from a given state may lead to many possible next states. Following all possible outcomes into the future leads to potentially exponential growth in the number of possibilities. It is impractical and incomprehensible to display all of that information at once. The screenshots given here are of an initial interface design for plan display that is intended to minimize the cognitive load of the user.[1]

This paper is organized as follows. We first present POET, the programmable online elicitation tool for preferences, and discuss the underlying representation of preferences and utilities. We present PlanScan, the plan displayer. We then walk through a decision process based on these two tools (assuming factored MDP model of a system and an appropriate planner). Finally, we discuss a smattering of related work.

2 POET, the Programmable Online Elicitation Tool

We developed POET to elicit undergraduate's preferences about their academic programs. Our assumptions were that some students do not know their preferences, and preferences may be contradictory, and may change.

*Department of Computer Science, † Department of Psychology, ‡ Department of Educational Policy, § Department of Curriculum and Instruction, University of Kentucky. Corresponding author email: goldsmit@cs.uky.edu. Work partially supported by NSF grant ITR-0325063.

[1] According to the Wikipedia [14], "Cognitive load is how much of a person's attention is being used up. For example, a person in a boring lecture will likely have a lower cognitive load than a soldier in battle."

In order to guide students, we introduced the notion of "archetypes". For instance, for computer science majors, we defined archetypical preferences for "Wants to go to grad school," "Wants to work in industry," and "Wants to party like a rock star." The latter archetype included strong preferences against morning classes, classes with large programming assignments, and against or for certain professors. The archetypes are initial sets of preferences that the students could agree with, or could adjust as desired. Archetypes are important because they present users with a example that they can then critique and adjust. Studies have shown that example-based interfaces can more accurately elicit user preferences and increase user confidence in their choices [7].

In order to allow contradictory preferences ("must graduate as soon as possible, while only taking Tues/Thurs classes between noon and 2pm"), we assumed an additive utility function that could assign both positive and negative values to the same instance. We offered both hard constraints ("must have" and "must not have") on attribute values, and soft constraints, which translated into utility.

We address the third assumption, that preferences change, by re-elicitation and replanning, and in particular by the interactions offered by our current suite.

2.1 POET Representation of Preferences

We have defined the following model for preference representation [10]. The universe over which preferences are expressed is a collection $A = \{A_1, \ldots, A_n\}$ of *preference attributes*. With each attribute A_i, we associate a finite domain $dom(A_i)$ of possible values.

The preference over one attribute, for instance, the professor, may depend on the values of other attributes, such as the course topic. We represent such a dependency as a tree with (in this case) PROFESSOR at the root and utilities at the leaves. Each path is interpreted as the conjunction of attribute values; each internal node is an attribute name and outgoing edges represent values for that attribute. Each attribute can appear at most once per branch, and each conjunction can appear at most once (as a branch or subset) within the preference forest.

We use a Likert scale from "MUST NOT HAVE" to "MUST HAVE", with weak and strong preferences for and against, and indifference also available. The extreme values represent hard constraints while values in between are soft constraints. We hide the numeric values of these preferences (their translation into utilities) from the user. The independence of the trees allows a user to express both positive and negative feelings about attribute values of the same instance. The software assumes a default value of "indifference" for those attributes and values that the user does not explicitly consider.

Each tree branch has a utility, and each state in the MDP satisfies at most one branch per tree. If the state satisfies no branches of the tree, then that tree contributes the indifference value, usually 0, to the overall utility of the state. The leaf values of all branches that are satisfied by the state are added together.

At present, we assign utility values on a linear scale, with "don't care" translating into a utility of 0, and positive utilities ("weakly prefer" and "strongly prefer") mapped to 1 and 2. Negative utilities are mapped to -1 and -2. We will experiment with other mappings in later work.

Because the trees function independently of each other, software can evaluate them in parallel. Furthermore, the evaluation of each tree on a given instance is fast.

2.2 POET Elicitation Process

The first step of the POET elicitation process, after login and tutorials, is the choice of an "archetype," or general description, which is linked to a set of suggested preferences. POET displays these preferences in tabbed panes labeled by attribute.

Once the user has selected an archetype, they are free to change anything presented in the preference structures: They may add new attributes or dependencies, delete attributes or dependencies about they indifferent about, or adjust the slider value for any dependency chain or single attribute. The archetypes are starting points and suggestions. The user may choose to jump immediately from the provided preferences to the planner and plan displayer, or they may do so at any later point in the elicitation process.

When the user requests a plan, the plan pops up a persistent window (as described in Section 3). Included in the window is a description of the preferences used to compute that suggestion. The window persists as is, even if the user then modifies their current preferences. This presentation allows the user to compare different outcomes and recover the preferences that led to their favorite recommendations. The user can access multiple recommendation windows as tabbed panes, making navigation easier and minimizing clutter on a potentially small screen.

3 PlanScan

The input to PlanScan is a model (a factored MDP description) and a plan. It maintains a link to the POET preference description for that plan. The plan maps states to actions. Each factored state is an assignment of values to the set of *state variables*, and each state has a set of allowable actions. Each action results in stochastic effects: the model specifies the probability distribution of winding up in a specific state.

PlanScan parses and displays the input plan. It uses a layout that the user doesn't adjust. Its layout flows from left to right, detailing the current state, suggested and allowable actions, possible consequent states, and the details of the selected consequent state.The GUI details the current state in a vertical, scrollable list of all attributes and their values. The input to PlanScan that describes the domain includes plain-text descriptions of attribute values for display purposes.

PlanScan displays actions in a series of labeled buttons. The first button shows the action that the plan recommends, and its text label is shown in boldface. The user may choose an alternative action from the list in order to explore a different trajectory. Based on the input file, PlanScan uses text-based names for actions as well as state attributes, and provides descriptions of those actions when the mouse cursor hovers over the button.

Once the user selects an action, PlanScan displays the set of possible consequent states. Rather than displaying multiple detailed state views, it generates a series of rectangles

of various sizes and colors in a horizontally scrollable pane. Each rectangle represents a possible consequent state. The sizes of the rectangles correspond to the likelihood of entering that particular state after taking the selected action; larger rectangles correspond to higher probabilities. The colors of the rectangles correspond to the utility (hence, the desirability) of each consequent state. Colors range from dark blue (very positive utility) through white (neutral) to dark red (very negative utility). The rectangles (states) can be ordered in decreasing order by probability or utility.

When the user selects a rectangle, PlanScan details the consequent state in the righthand pane in the same format in which it shows the current state. Attribute values in the current and result state are shown in boldface if they differ. Once the user has selected a consequent state, clicking the "forward" button advances to that state. The "forward" button shows a right-facing arrow; the "back" button, which lets the user restore the previous state from a stack, shows a left-facing arrow. Together, these buttons mimic the advance and retreat functions of popular web and file browsers.

When exploring a plan, a user might want to compare possible outcomes after a series of actions. PlanScan allows users to chain together a series of actions (which we call *action series*) and look at the possible outcomes after taking that series of actions, while maintaining the current state.

Action series allow users to explore future states without backtracking and trying all the trajectories manually; the current state provides a static point of reference. This facility allows users to "jump" as well as "walk" through a plan. Users can utilize action series to focus on fragments of the plan that are of interest to them.

In many decision-support applications, one can define subgoals. A subgoal is either a complete state or a partial list of attribute values. When a user defines subgoals, PlanScan automatically adds a new tabbed window to display all subgoals being tracked. For each tracked subgoal, PlanScan shows the probability of reaching that subgoal by following the plan from the current state and the expected number of actions to do so. The user may add or remove subgoals at any point; PlanScan does not limit the number of tracked subgoals.

4 A Walkthrough

We illustrate how the user interacts with POET and PlanScan on the following example (Figures 1–4). Petra is a junior in the Department of Computer Science. She is considering which courses she should take in the next semester. Because Petra likes computer gaming and considers programming computer games to be a possible career path, she decides that she is very interested in taking an AI class with Dr. Goldsmith. At the same time, she has heard that she would be expected to speak at every class in a programming languages course taught by Dr. Finkel. Petra does not like this prospect. Using POET, Petra specifies her preferences for AI (high) and programming languages (very low) (Figure 1). This information is passed to the planner, which constructs a tentative plan for the remaining three semesters of Petra's study. The plan is passed to PlanScan, which displays it for Petra (Figure 2).

While walking through the plan, Petra and her advisor notice that the plan estimates the probability of success (getting a grade of "B" or higher) in the AI class to be rather low. This outcome can be explained: Petra got "C"s in Discrete Math and Logic courses, and in the minds of the advisors who built the model used by the planner, performance in AI is closely related to performance in these classes. During the planning stage, Petra's high utility of taking the AI class "overrode" the low probability of success.

Petra is somewhat discouraged by the advisor's comment that the AI course might be hard for her to succeed in. The advisor proposes alternatives to the AI course, including programming languages. As it turns out, Petra did very well in the prerequisite classes for this class, so the chances of success in it appear to be significantly higher. Because of the low initial utility, the possibility of taking programming languages had not been suggested.

Realizing she is better off with an "easier" programming languages class, Petra mentally downgrades the inconvenience of oral presentations. She upgrades her preference for this class in POET and decreases her preference for the AI class (Figure 3). The new plan is computed (Figure 4). In this plan, the suggested course of actions is to take the programming languages class.

In this simplified example, the "straw plan" based on the student's initial description of her preferences gave her a concrete basis for preference revision. By linking preference elicitation, planning, and plan display, we are able to involve the user in the planning process in an interactive manner. We conjecture that this interaction leads to better plans and happier users.

5 Related Work

The area of interface design for MDP-based decision-support systems is a new one, with limited literature. There is a wider body of work on interfaces for recommender systems, which we do not survey here. Recommender systems require potentially complex elicitation (direct or indirect) of a user's preferences, but the actual recommendations are reasonably straightforward to present.

Similarly, we do not discuss the slightly more relevant work on interfaces for deterministic planners, such as those used in the Rochester Interactive Planning System [3]. However, we look briefly at some of the cognitive psychology work on interface design as it relates to our task of preference and policy presentations.

The interface for a decision support system must take human processing into account. Lin, Choong, and Salvendy [5] examine some key areas of importance in interface design. Several of their categories play into the idea of representing a strategy planned using decision theoretic planning. These include compatibility, consistency, flexibility, minimal actions, minimal memory loads, perceptual limitations and user guidance (defined below). The GUI design should consider each category, so that the interface shows all necessary information for the user to complete their part of the plan, and displays that information in a manner easily understood by the user.

The interface must be able to display any output of the planner (compatibility), and must maintain a consistent pre-

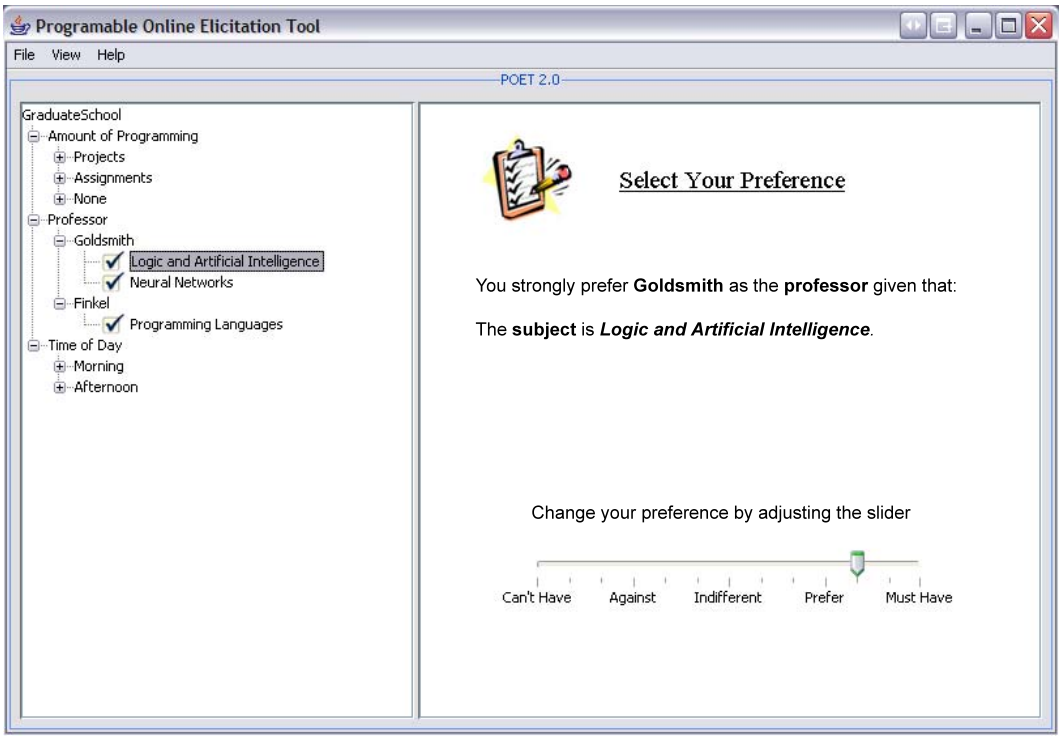

Figure 1: POET: Initial preferences

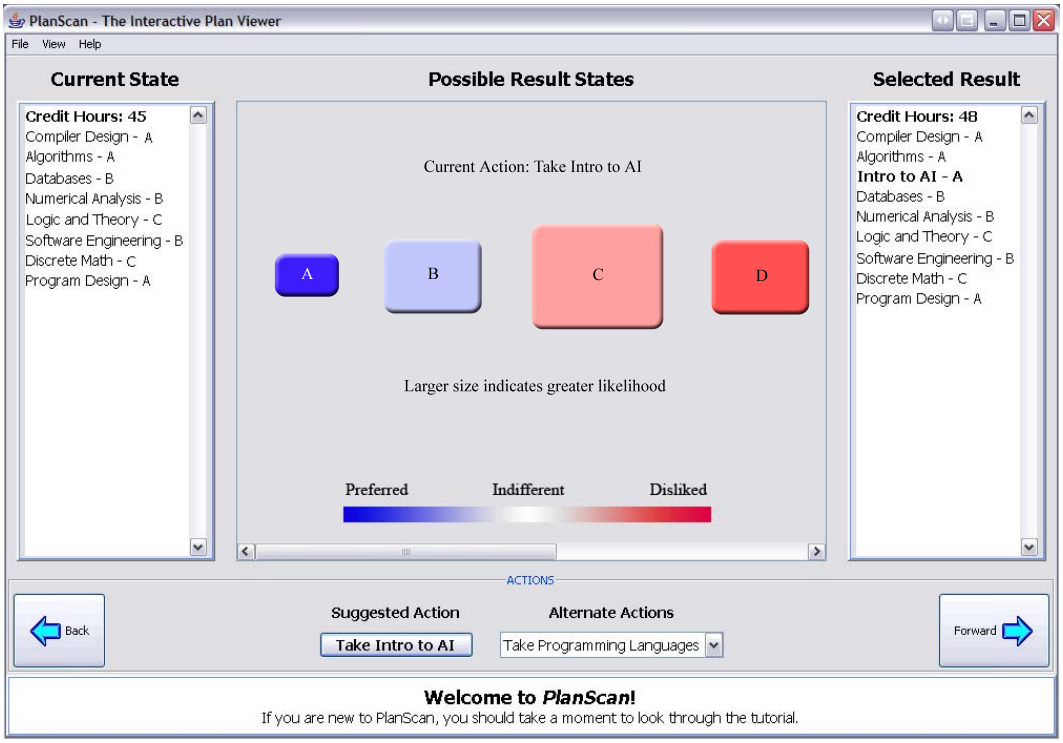

Figure 2: PlanScan: Initial plan

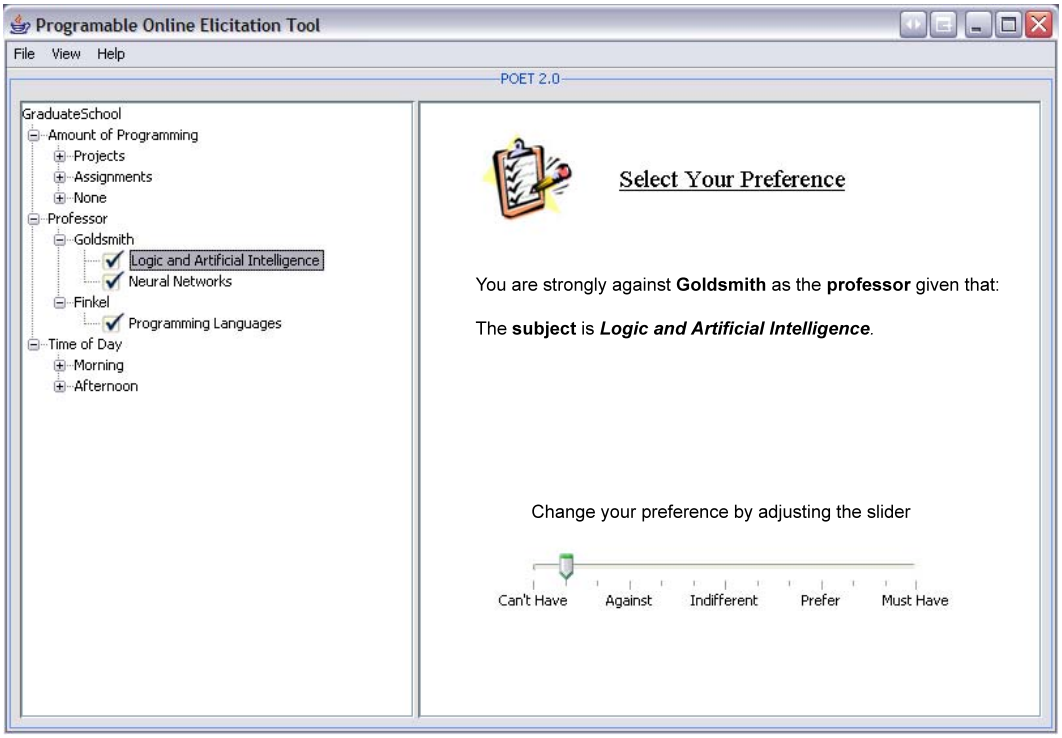

Figure 3: POET: revised preferences

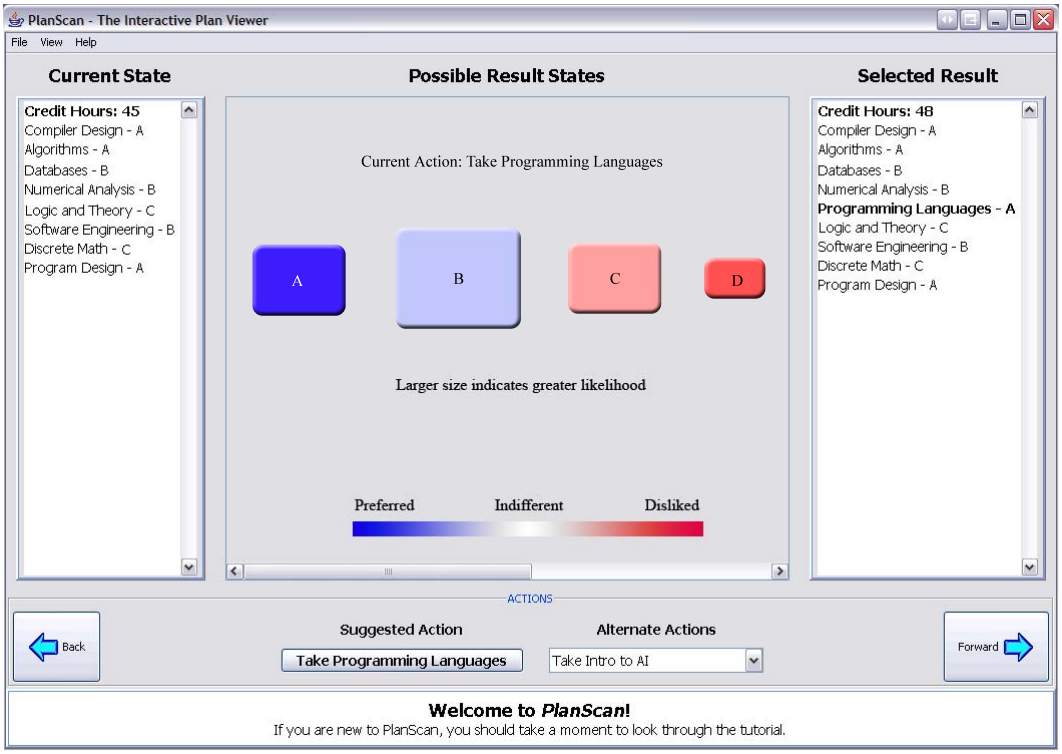

Figure 4: PlanScan: revised plan

sentation across different outputs. Flexibility refers to the interface's ability to handle varying output formats. It is important that the user be able to both comprehend the presented plan and be able to act on it. Consistency is one aspect of comprehensibility, as is minimal memory load: The user should not be expected to memorize a complex manual in order to use the display tools or to understand what is displayed. Minimal actions refer to the actions required of the human in order to get the desired output.

Ullman [13] claims that users often choose a method based on cognitive limitations. A display should be easy to understand, with all portions clearly labeled; users should not have to spend time contemplating their next move. Multiple paths should be presented in a way that allows users to see the pros and cons of the plan before taking an action, giving the user further flexibility in their choice of actions.

One of the few plan displayers we found was particular to the domain of shopping in an airport on the way from check-in to boarding [1]. However, the displays do not show multiple plan trajectories, nor do they exhibit a "you are here" on the trajectories.

One of the eight fundamental rules of interface design [12] is to limit the amount of short-term memory the user must dedicate to the interface. One can achieve this goal with a full, clear path that gives users a sense of their progress toward a goal (whether it be maneuvering through an airport, or, in the academic domain, how much longer until graduation. While MDP policies are good at answering "What should I do next?", a raw representation of a plan does not show progress toward goals or subgoals.

In contrast with the airport planner, PlanScan graphically and vividly indicates when there are multiple possible futures. It allows the user to explore one or more of futures, giving them a measure of control over the exploration process. The addition of a subgoals allows the user to track their progress toward the goal as they explore the plan. The ability to choose actions different from those suggested by a plan gives the user further control over their environment.

Finally, choosing other actions, insisting on those actions, or by altering preferences in a way that pushes desired actions into the plan, puts the user into the decision-making loop as a partner rather than a servant to the computer.

There is considerably more literature on interfaces and interaction between preference elicitation and solution displays for recommender systems. We cannot begin to do justice to that subject here, but we mention particular work that has influenced our thinking.

Faltings, Pu, and others [4; 8] have investigated eliciting preferences by *example critiquing*. They display ranked results of a search and ask users whether they prefer the top-ranked items, and if not, why not. Example critiquing is meant to avoid traditional preference elicitation, which can be both complex and insufficient to articulate the users' actual preferences. It allows users to state their preferences on any attribute and in any order. Their work supports explicit tradeoffs between attribute preferences, but not explicit conditional preferences.

Our work uses utilities to implicitly represent tradeoffs, and explicitly represents conditional preferences. The similarity with Faltings, et al.,'s work is our use of candidate critique to refine the user's understanding of their own preferences. It is this interaction between MDP planner, plan displayer, and preference elicitor that makes our work unusual and exciting.

References

[1] T. Bohenberger, A. Jameson, A. Kruger, and A. Butz. Location-aware shopping assistance: Evaluation of a decision theoretic approach. In *Proc. 4th International Symposium on Human-Computer Interaction with Mobile Devices*, pages 155–169, 2002.

[2] C. Cornalba, R. Bellazzi, S. Quaglini, R. Bellazzi, and M. Stefanelli. An adaptive decision support system for risk management in haemodialysis departments. Technical Report SS-05-02, AAAI, 2005.

[3] M. O. Dzikovska, J. F. Allen, and M. D. Swift. Integrating linguistic and domain knowledge for spoken dialogue systems in multiple domains. In *IJCAI '03*, 2003.

[4] B. Faltings, M. Torrens, and P. Pu. Solution generation with qualitative models of preferences. *Computational Intelligence*, 20(2):246–263, May 2004.

[5] H. Lin, Y. Choong, and G. Salvendy. A proposed index of usability: a method of comparing the relative usability of different software systems. *Behavior and Information Technology*, 16:267–278, 1997.

[6] Merriam-Webster. *Webster's 9th New Collegiate Dictionary*. 1992.

[7] P. Pu and L. Chen. Integrating tradeoff support in product search tools for e-commerce sites. In *Proceedings of 6th ACM Conference on Electronic Commerce*, 2005.

[8] P. Pu and B. Faltings. Decision tradeoff using example-critiquing and constraint programming. *Constraints*, 9(4), 2004.

[9] S. Quaglini, T. Giorgino, C. Rognoni, M. Stefanelli, E. Marchesi, and J. Baccheschi. Adapting an hypertension home-monitoring system to the patient's cardiovascular risk. Technical Report SS-05-02, AAAI, 2005.

[10] J. Royalty, R. Holland, J. Goldsmith, and A. Dekhtyar. POET, the online preference elicitation tool. In *Proc. AAAI Workshop on Preferences in AI and CP: A Symbolic Approach*, 2002.

[11] A. Sboner, R. Bellazzi, P. Carlid, and M. Cristofolinie. User-tailored clinical decision support systems. Technical Report SS-05-02, AAAI, 2005.

[12] B. Shneiderman. *Designing the user interface: strategies for effective human computer interaction*. Addison Weseley Longman, Inc., 3rd edition, 1998.

[13] D. G. Ullman. The ideal engineering support system. *Research in Engineering Design*, 13:55–65, 2000.

[14] Wikipedia. Cognitive load, March 2005.