POET: The Online Preference Elicitation Tool*

James Royalty, Robert Holland, Judy Goldsmith, Alex Dekhtyar
Department of Computer Science
University of Kentucky
Lexington, KY 40506
jroya00@cs.uky.edu, robert@macauley.com,
{goldsmit,dekhtyar}@cs.uky.edu

Abstract

If the goal of eliciting accurate user preferences over a complex value space is to be realized, the elicitation process must be through while at the same time not overly burdensome to the user. An ideal preference elicitation tool would maximize the information about preferences acquired from the user while minimizing the number of queries required. To this end, we present POET: the Online Preference Elicitation Tool.

POET is a graphical Java applet designed to elicit complex preference structures to form a *utility function*. Once the elicitation process is complete POET outputs the resulting utility function as an XML document. Internally, POET represents utility for independent and dependent values as a set of *dependency trees*. Though POET's output and internal representation involves real numbers, from the user's point-of-view, the experience is completely symbolic.

Introduction

Because of the ever-expanding body of human knowledge and the increasing demands of society and of the modern workplace, the expectations of a college education are steadily increasing in complexity. This often translates into complex university and departmental requirements for college graduation. It is the job of faculty, or designated advisors, to help university students navigate these requirements.

Unfortunately, in many academic institutions, these advising duties come on top of increased pressure on faculty to: improve their teaching and their department's retention of students, increase research and funding thereof, and most of all, increase the flow of administrative paperwork. All of these demands, including those inherent in the (sometimes Byzantine) requirements for students, take away from the human interaction component of advising. Thus, there are clear benefits to automating the mechanical parts of academic advising including: schedule consistency checking, progress toward degree completion, and optimization of the student's educational priorities.

We hope, by automating the advising process, to increase the time that advisors can spend actually listening to their advisees and helping these students prioritize goals and optimize their time at the university. The Bayesian Advisor tool (Dekhtyar *et al.* 2001), of which the work described in this paper is a part, aims to do just that. The advisor tool will check whether a student is on track to meet requirements in a way consistent with their priorities; it will suggest courses to further that process, and will be able to compare different plans (from disparate sources) and recommend those with the highest expectation of success for that student.

Ideally, success will be defined by each student according to her priorities. Those priorities could include: rapid progress toward graduation, a high GPA, a concentration of courses in a sub area, a concentration of courses by their favorite professors or at their favorite times of day. In short, to choose a sequence of courses that optimizes the expected utility of the student's academic career, it is necessary to determine that individual's preferences, priorities, and the corresponding utility function. In this paper, we describe a representation of, and a procedure for, eliciting such preferences and forming them into a utility function suitable for use by the Bayesian Advisor tools planner.

The work described here is specific to the academic domain. To the best of our knowledge, we are the only group looking at modeling academic advising using Bayes networks. There is, however, significant work on using Bayes nets in other educational settings. For instance, (Nicholson *et al.* 2001) describes a tutoring system for elementary mathematics based on a Bayes network model of student knowledge. While the high-level goals are similar (namely to improve student learning/utility) there are considerable differences in these goals and projects. To begin with, the tutoring systems are inferential, whereas the Bayesian Advisor tool is decision-theoretic. Furthermore, tutoring systems focus on actual learning while the Advisor focuses on whatever goals the student provides. These may include maximizing learning or optimizing exam performance but are neither necessary nor necessarily sufficient.

We expect that our careful attention to the details of this domain will yield a better understanding of, and usable software for, the preference elicitation task for

*A little poetic license.

planning problems in general. We begin by giving a more detailed overview of the application domain of the Bayesian Advisor tool and the role of preferences therein. We then discuss the representation we have chosen for preferences and its pros and cons. Next, we describe the actual elicitation process and the software tool we are developing to support it. We then compare our approach with those of some other recent preference representations and elicitation processes. Finally, we discuss probable extensions of this work.

Bayesian Advisor Background

The underlying mechanism of the Bayesian Advisor tool will be planning over a Bayes network. The network will represent the causal dependencies of success in courses based on success in other courses or additional factors such as exam scores. For instance, the network might represent success in an introductory AI course as depending on success in courses on algorithms, programming, and logic.

We separate random variables we plan to include in the model into two basic categories: *performance in university courses* and *supplemental information*. Student performance in each university course will be a random variable, and therefore, a node in the network. Supplemental information or overall predictors such as current grade point average (GPA), high school GPA, as well as scores on standardized tests (SAT, ACT, GRE, TOEFL, and the like) will also be nodes. Besides these *observable* variables, we may, whenever possible, represent some of the *hidden* variables present like "mathematical sophistication", "programming skills", "analytical ability", and so forth.

The inputs to the system will consist of a student transcript which can be viewed as an instantiation of some of the variables of the network modeling the university curriculum, and a utility function for the student in question. A planning algorithm will be employed to generate recommendations for subsequent courses. The utility function, elicited as described in this paper, will be used by the planning algorithm to compare candidate plans; the plan(s) with higher utility value are considered more valuable to the user.

Note that probabilities will be used to generate predictions on student performance *for use in internal computations only*. The Advisor will not publish these predictions to the student, faculty, or advisors; doing so might affect both the student's performance and the professor's grading of that student.

The Online Preference Elicitation Tool

The Online Preference Elicitation Tool (POET) is being developed in the context of the Bayesian Advisor tool. Input to and output from POET will be in an XML format consistent with the Semistructured Probabilistic Object Markup Language (Zhao, Dekhtyar, & Goldsmith 2002) used by other parts of the Advisor. Planning algorithms for this system will use utility functions generated by POET.

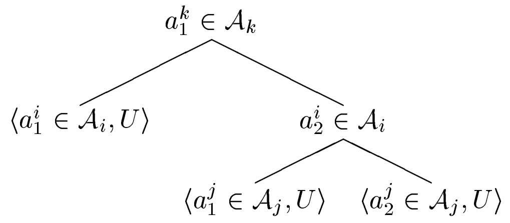

Figure 1: Dependency tree for values from the domains of A_k, A_i and A_j. U represents the utility, provided by the user, for the given leaf.

Representation

Preferences We define $A = \{A_1, A_2, \cdots, A_n\}$ as a set of *attributes* over which the user has some preference. Each A_i can take on some *value* from the set $\mathcal{A}_i = \{a_1^i, a_2^i, \cdots, a_n^i\} \equiv dom(A_i)$ of possible values; this is the domain of A_i. For instance, in academic advising, we might have $A = \{Professor, Subject\}$ with $\mathcal{A}_{Professor} = \{Goldsmith, Marek\}$ and $\mathcal{A}_{Subject} = \{Databases, AI\}$.

We require the user indicate her preference for values of each $\mathcal{A}_i$, and any combination $\mathcal{A}_i \times \cdots \times \mathcal{A}_j$ she deems relevant. Preferences for individual values or combinations of values are represented as real-valued utilities in a fixed, finite range. Values for which the user is indifferent are given the special designation λ.

Initially, we assume all preferences on values are *independent* unless the user states some dependency structure during elicitation, and, that the user is *indifferent* to all values not explicitly evaluated – in other words, those values have a utility of λ.

Dependency Trees Our goal is to represent (explicitly or implicitly) student preferences for each value of each attribute. Whether dependent or independent, preferences are represented as sets of decision trees, which we term *dependency trees*. The utility function for each tree assigns a numerical value to each path; the overall utility function is the sum of the values on all the trees.

A leaf in a dependency tree consists of an attribute value and a utility value. The utility value applies to the path represented by that leaf. Trees representing independent values consist of a single leaf (a tree of *height* 0). Trees for dependent values are more complex. Figure 1 depicts a dependency tree for dependent values. The value a_1^k depends on a_1^i and a_2^i. a_2^i, in turn, depends on a_1^j and a_2^j.

Internal nodes have as children a set of values from the domain of some particular attribute A_i. That set does not need to include all possible values from A_i; those values not included are given the designation λ. No attribute value appears as an ancestor (or descendant) of itself on any tree.

A sample dependency tree is presented in Figure 2. This tree represents a student's utility for the *MathematicalSophistication* value from the *AncillarySkills* attribute. According to the tree, the student believes that mathematical sophistication can be obtained by taking either an *AI* or a *Database* class. However, the utility of this skill differs based on who teaches the class. If the AI class is taught by *Goldsmith* the utility depends on the time of day when the class is offered; if it is taught by *Marek*, it depends on the student's expected courseload.

Utility Function The overall utility function will be the sum of the utilities of the values for each attribute. Note that an attribute may occur in several trees. A student might mildly prefer morning classes but consider Marek's morning class of greater utility than Goldsmith's. The utility of a given system state (a static transcript) and an action is the sum of the utilities for the relevant attribute.

We define the utility function u from values to dependency trees. Each tree will assign a value from $\mathcal{A}$ to each path in the tree. Trees with one node have one path, thus assigning a number to that value; trees with no nodes implicitly assign λ to the corresponding value.

Reasoning with the Utility Function

The goal of the Bayesian Advisor tool is to produce relevant plans of study for students. Each plan is a function that finds actions (courses to take) for each state of the system.

Plans can be evaluated by considering each possible state of the system, or *instance*, that is reachable from the start state, and taking the sum of these values, weighted by the probabilities of reaching each instance. Because our utility function is factored, and the instances are factored, this can be done efficiently.

The value of an instance, i.e., a partial transcript, is the sum of the values of the individual components, namely courses, according to the utility function. Each component can be evaluated in terms of each attribute: *Subject*, *Professor*, *TimeOfDay*, *AncillarySkills*, *GPA*, and so on. The student's utility for each value can be looked up quickly using an adequate hash function on the (attribute) values themselves. Those values that are part of a dependency tree (either root or node) can be cross-referenced, creating a data structure of pending evaluations. In the case where there are no dependency trees, or at least no trees indicating concurrent-type dependencies ("Database courses are desirable if not concurrent with Networks courses", for example), the evaluation of fluents can be done in parallel with no communication. If there are concurrency-type constraints, then parallel evaluation will require two passes.

Elicitation Process in POET

It is necessary that the elicitation process be thorough, while at the same time, not burdensome to the user. In other words, we wish to maximize the information about preferences acquired from the user while minimizing the number of queries required.

The academic domain, like other complex domains, such as military modeling (Laskey & Mahoney 2000), has attributes with many values. Consider, for instance, the attributes of *Professor* and *MilitaryVehicle*, respectively. Eliciting preferences for each value (and combination of values) is potentially mind-numbing. Thus, our challenge is to acquire many preferences implicitly, using a reasonably small number of queries.

POET Features

Our approach to making the elicitation process more tractable includes the following:

- archetype selection;
- preset utility on values from the archetype;
- simple utility specification interface;
- intuitive dependency specification.

We define an archetype to be an easily recognizable summary of a typical student's goals and priorities. An archetype consists of a set of preselected attributes $A' \subseteq A$, along with a set of values $\mathcal{A}'_i \subseteq dom(A'_i)$ from each attribute in A' deemed relevant to the given archetype. Some values will be preset to some utility if they are believed to be of specific importance to the particular archetype. The remaining values will be set to λ.

In order to further simplify the elicitation process, some values from each $\mathcal{A}'_i$ will be omitted altogether[1]. Presenting the user with these archetypes allows us to "jump start" the elicitation process in exchange for a minimal amount of user effort.

At present, assumptions about attributes contained within each archetype, and any preset utility, are based on information informally elicited from experts (namely students and advisors) at the University of Kentucky.

Interacting with POET

A user's interaction with POET proceeds according to the following five stages:

1. explanation of the process;
2. archetype selection;
3. specification of utility for independent values;
4. specification of structure for dependent values and the associated utility;
5. result presentation.

Archetype Selection The elicitation process begins with a simple, friendly, user-oriented explanation of the above stages. After choosing to continue, the user is prompted to select an archetype that best fits her goals and priorities. Example archetypes used in POET include:

- Work in the computer industry
- Go to graduate school

[1] The user will have the opportunity to add or remove (hide) any attribute or value, perhaps subject to constraints.

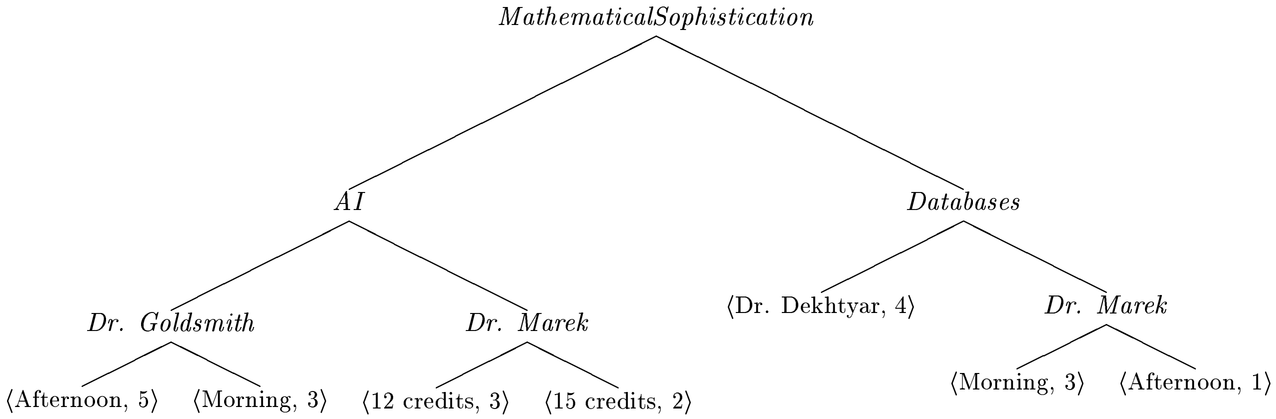

Figure 2: AI with Goldsmith in the morning or databases with Marek in the afternoon?

- Party like a rock star
- Graduate with a high GPA
- Graduate as soon as possible
- I don't know[2]

Currently the choice of archetype is mutually exclusive: the user can select only one of the above options. The feasibility (indeed the desirability) of allowing the user to select multiple archetypes, when applicable, is a possible future extension; see the last section for more information.

Utility for Independent Values Once an archetype has been selected the user is given the opportunity to evaluate each value according to her preference, using slider bars (see Figure 3.)

For instance, $TimeOfDay$ is relevant to both the would-be rock star and the industry-bound student. However, one would expect the rocker to strongly prefer not to have morning classes while the budding member of the workforce should start learning to function properly before noon.

If the user believes certain attributes or values should be added (and/or removed) to (from) her archetype, she will be given the opportunity to make such changes at this point.

Utility for Dependent Values Adjacent to each slider bar is a checkbox labeled "It Depends". If this box is checked, the slider value is ignored and the user is asked to specify upon which *attribute(s)* her preference for the related value depends (see Figure 4.)

Once the dependent attributes are chosen, the values for each attribute are displayed *in combination* and utilities are elicited for each in the same fashion described previously. In addition, for each value combination, the user is given the opportunity to specify dependencies using the "It Depends" semantics described previously.

[2]At present, this archetype includes all attributes from A, e.g. $A' \equiv A$.

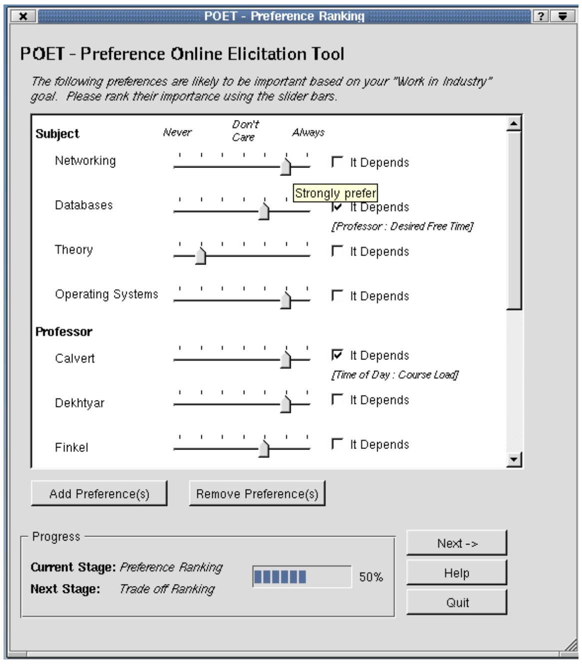

Figure 3: Evaluating independent features in POET

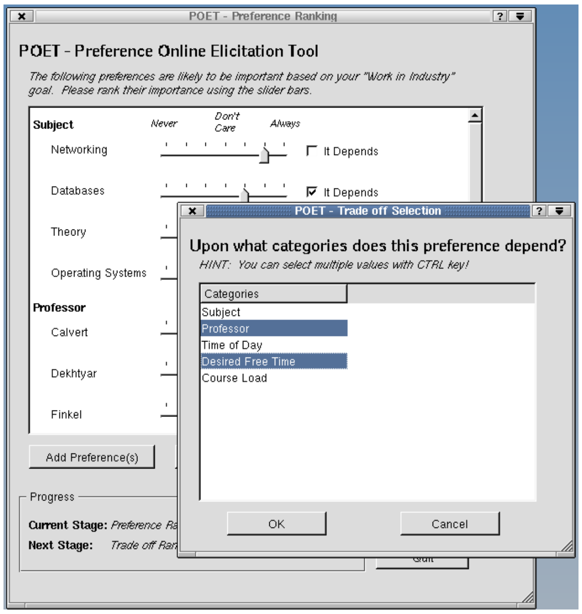

Figure 4: Specifying a dependency structure in POET

Care must be taken to prevent the user from specifying cyclic dependencies – POET enforces this restriction.

Result Presentation Finally, after the dependency specification and evaluation phase, POET will present the user with a "friendly" summary of the preference information elicited from her, as a simple means of verification. This could take the form of programmatically generated English sentences such as, "You are very interested in taking AI classes. You are not interested in taking Networking in the morning, but are indifferent to Networking in the afternoon." Results could also be presented graphically as actual dependency trees similar to the one shown in Figure 2, or a histogram depicting the utility value contained in the leaves of each tree.

As a result of this presentation, the user may wish to revise her preferences – she will be given that opportunity here.

Once the user verifies her results, POET will encode the preference information (in the form of a utility function) in an XML document for use in other Advisor tools.

A Simple Example

Consider a hypothetical undergraduate, Sue Smarty. Her educational goals are to graduate in at most nine semesters with a degree in computer science and a GPA of at least 3.9. After graduation Ms. Smarty would like to work in the computer industry, particularly in the areas of networking and databases. Along the way, she would prefer to have some classes in the morning[3] and would like to avoid mid-afternoon classes, if possible, as she tends toward involuntary naps.

During an advising session Ms. Smarty is faced with a choice: she can take a late-afternoon course from her favorite female professor in computer science, who is known to be very challenging, or she can take a much easier mid-afternoon course from a male professor. Which should she choose?

Ms. Smarty's decisions may be quite simple: if her overriding priority is a high GPA, then she takes the easier course. A more complex situation arises if she has trade-offs (in other words, a dependency structure) in mind: time is more important than the professor's gender, but the likelihood of a high grade is more important than either.

When Ms. Smarty starts using POET, she will give very high utility to the value $HighGPA$ within the GPA attribute. Within the attribute of $Professor$, when choosing her favorite female professor, she will click "It Depends" and then choose the GPA and $TimeOfDay$ attributes. Figure 5 depicts a possible set of dependency trees for Ms. Smarty.

Related Work

There are a variety of approaches in the literature to the problem of representing and eliciting preferences. We give a brief survey of some of them here.

A significant body of work on qualitative preferences exists to date. These are used for applications such as product database searches (see the discussion in (Boutilier *et al.* 1999) and the implementation of the Active Buyer's Guide website (Active Decisions, Inc.)) and constraint-based product configuration. Because we intend to use our preference information in decision-theoretic planning, we require not only a ranking of preferences but actual utility *functions.*

From the qualitative preference literature we surveyed, the representation most similar to ours is the CP network (Boutilier *et al.* 1997). A CP network is a graphical representation of qualitative preference structures. Our dependency trees are similarly graphical in nature. However, there are two significant differences. First, our trees not only provide an ordering on preferences, they also form a utility function from attribute values to numerical values. Second, our trees are perhaps simpler than CP networks in that they (the trees) do not require nodes to be annotated with tables. On the other hand, a CP network is a potentially more compact representation in that a single network encodes relations among all values, whereas we require an individual tree per value/dependency structure[4].

The problem of capturing preferences has also been addressed recently by means of collaborative filtering techniques (Breese, Heckerman, & Kadie 1998). Collaborative filtering works by comparing the behavior of

[3] After all, she has to learn to wake up early *at some point.*

[4] We should also mention that the "depends on" phase of our elicitation process was inspired by the trade-off phase of CP network-type elicitation (Boutilier *et al.* 1999).

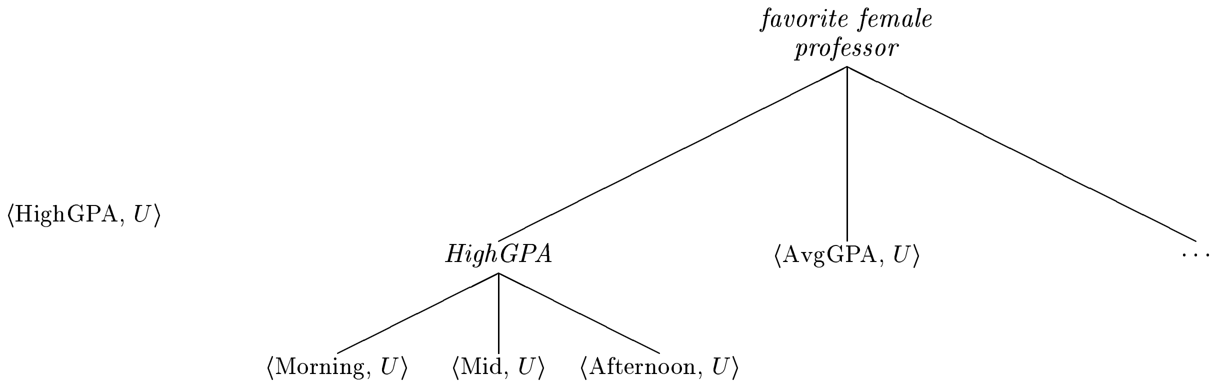

Figure 5: Possible set of dependency trees for Sue Smarty. U represents the utility for the given leaf. Note the tree to the left representing an independent $HighGPA$ value.

the current user with a database of previous uses of the same system and extrapolating possible future behavior of the user. For this approach to work, one must have a large enough collection of preferences from previous users. Currently, in our project, this is not feasible. However, once the preference elicitation system is deployed, we may be able to accumulate an anonymous database of preferences. We would then be able to update our user archetypes by using clustering methods, perhaps along the lines of (Jain, Murty, & Flynn 1999).

One can also represent preferences as feature vectors over a discrete or continuous domain. This is used in the FindMe system (Burke, Hammond, & Young 1996), and probably in various web-based product selection systems. This representation does not allow for implicit or succinct representation of complex or dependent preferences.

On the other hand, one can use feature vectors over continuous space to represent preference functions (Keeney & Raiffa 1976). This allows the application of analytic methods to find dominating preferences. However, the major disadvantage becomes apparent when one considers elicitation: it could take many queries to approximate a smooth curve.

There is also a large body of work on learning user preferences passively, (non-interactively.) This work supports customized web crawling and targeted advertising. It is singularly unsuitable for the academic advising domain. By the time a system has acquired enough data on a user, that user has almost certainly acquired sufficient credits to graduate! However, one might be able to apply this process to a database of complete transcripts to infer and then classify utility functions for use in classification-based utility elicitation (Chajewska *et al.* 1998).

Future Work

Work on POET is ongoing; we envision the addition of several features in the coming months. Areas to receive attention are presented below.

Extensibility Given that POET is a component of the Bayesian Advisor tool, it must output a utility function encoded in XML (as XML is used for data transfer between different parts of the Advisor tool). This XML-encoded utility function will be, in turn, passed downstream to the planning tool.

We have two parallel goals for the future development of POET:

- build the best tool for this domain, and
- build a *general* preference elicitation tool for use with other planning algorithms/tools.

As mentioned previously, POET is domain specific; however, we expect that careful attention to our domain will enable us to produce a tool useful for preference elicitation in other domains.

We are developing XML schemata for representing preferences and utility functions that will be generic enough to handle diverse application domains. In the future, POET will construct generic GUIs for preference elicitation (on the fly) given an XML instance document describing a particular preference structure.

Archetypes It is not yet clear whether it would be efficacious to allow a user to select multiple archetypes, such as “Get MBA” and “Work in the computer industry.” Combinations like “Maximize GPA” and “Party like a rock star” do not seem to be very compatible.

If the selection of multiple archetypes were permitted certain constraints (expressed in the input XML instance document) should be in place to prevent situations like the latter. A friendly warning message might suffice or, in some cases, referral to a human advisor.

There are two directions in which the use of archetypes can be expanded. First, current archetypes are based solely on internal, informal, elicitation. As POET is deployed and used, input from students could be gathered in order to bring the archetypes in line with *their* self-images. Also, our current archetypes reflect only the goals/priorities of undergraduate computer science students. We would, in the short term, like to expand them to include other types, such as graduate students. In the long term, we envision that the notion of archetype could be extended to express generalities in other domains.

Constraints and Warnings There are numerous domain-specific constraints that could be included. For instance, during the "It Depends" phase of eliciting a student's utility for AI courses, POET could be constrained to only list those professors who actually teach or might teach AI classes. Encoding such information would significantly reduce the number of dependent values the user would have to evaluate.

Other domain-specific constraints include forbidden course combinations due to curriculum requirements, departmental prerequisites, and the prevention of overloaded schedules.

Verification We have not yet implemented a verification stage for the elicitation process. We expect to use a version of candidate critique (Linden, Hanks, & Lesh 1997) in the following manner: the user will be presented with a few precomputed and easily described plans, and asked to rank them. This ranking will be compared with the ranking generated by evaluating the plans according to the user's utility function. If the two rankings differ, the user will be prompted to refine the utility function.

Later versions may allow the user to specify a plan (or plans) for evaluation, and/or use plans generated by the planning tool. Thus, a user's own plan could be compared to the generic plan for her archetype. This might lead to updates either of the elicited utility function or of the user's self knowledge.

Acknowledgments

This research is partially supported by NSF grant CCR-0100040. Special thanks to Terran Lane[5] (*terran@ai.mit.edu*) for valuable feedback and assistance throughout this effort.

References

Active Decisions, Inc. Active Buyer's Guide (http://www.activebuyersguide.com).

Boutilier, C.; Brafman, R.; Geib, C.; and Poole, D. 1997. A constraint-based approach to preference elicitation and decision making. In Doyle, J., and Thomason, R. H., eds., *Working Papers of the AAAI Spring Symposium on Qualitative Preferences in Deliberation and Practical Reasoning*, 19–28. Menlo Park, California: American Association for Artificial Intelligence.

Boutilier, C.; Brafman, R. I.; Hoos, H. H.; and Poole, D. 1999. Reasoning with conditional ceteris paribus preference statements. In *Proc. Conference on Uncertainty in AI*, 71–80.

Breese, J. S.; Heckerman, D.; and Kadie, C. 1998. Empirical analysis of predictive algorithms for collaborative filtering. In *Proceedings of the 14th Conference on Uncertainty in Artificial Intelligence (UAI-98).*

Burke, R.; Hammond, K.; and Young, B. 1996. Knowledge-based navigation of complex information spaces. In *Proc. AAAI '96.*

Chajewska, U.; Getoor, L.; Norman, J.; and Shahar, Y. 1998. Utility elicitation as a classification problem. In Cooper, G. F., and Moral, S., eds., *Proceedings of the 14th Conference on Uncertainty in Artificial Intelligence (UAI-98)*, 79–88. San Francisco, California: Morgan Kaufmann.

Dekhtyar, A.; Goldsmith, J.; Li, H.; and Young, B. 2001. Bayesian Advisor Project I: Modeling academic advising. Technical report, University of Kentucky, Department of Computer Science, Lexington, Kentucky.

Jain, A. K.; Murty, M. N.; and Flynn, P. J. 1999. Data clustering: a review. *ACM Computing Surveys (CSUR)* 31(3):264–323.

Keeney, R. H., and Raiffa, H. 1976. *Decisions With Multiple Objectives: Preferences and Value Tradeoffs.* New York: John Wiley and Sons, Inc.

Laskey, K. B., and Mahoney, S. M. 2000. Network engeinnering for agile belief networks. *IEEE Transactions on Knowledge and Data Engineering* 12(4):487–497.

Linden, G.; Hanks, S.; and Lesh, N. 1997. Interactive assessment of user preference models: The automated travel assistant. In Jameson, A.; Paris, C.; and Tasso, C., eds., *Proceedings of the 6th International Conference on User Modeling (UM-97)*, CISM, 67–78. Springer.

Nicholson, A.; Boneh, T.; Wilkin, T.; Stacey, K.; Sonenberg, L.; and Steinle, V. 2001. A case study in knowledge elicitation and discovery in an intelligent tutoring application. In *Proc. Uncertainty in Artificial Intelligence.*

Zhao, W.; Dekhtyar, A.; and Goldsmith, J. 2002. A probabilistic database to manage CPTs (work in progress).

[5]Now with the University of New Mexico in Albuquerque (*http://www.cs.unm.edu/*).