Monte Carlo Preference Elicitation for Learning Additive Reward Functions

Stephanie Rosenthal and Manuela Veloso
Carnegie Mellon University
Pittsburgh, PA USA
srosenth,veloso@cs.cmu.edu

***Abstract*— AI agents including robots often use reward functions to evaluate tradeoffs between different states and actions and to determine optimal policies. We are particularly interested in reward functions that can be decomposed into an additive sum of subrewards that are computed on independent subproblems or features of the state space. If these subrewards capture different reward metrics, such as user satisfaction and task completion time, it is unclear how to scale the subrewards in the reward function to produce an appropriate policy. In this work, we propose and evaluate a novel Monte Carlo method for learning the scaling factors of subrewards, in which the training elicits humans' preferences between two state-action scenarios. Because the algorithm elicits preferences over explicit scenarios, it is less susceptible to human error than previous elicitation approaches. The preferences are used to generate a set of inequalities over the scaling factors that we solve efficiently using a linear program. We show that our algorithm asks for a number of preferences proportional to log of the number of scaling factor hypotheses used in the Monte Carlo method.**

I. INTRODUCTION

Many AI agents and robots make decisions about actions to take by assigning rewards to states and actions and choosing the actions that maximize the agents' reward. Defining the reward for each individual state,action pair can be tedious, particularly if the state space is large, and requires knowledge about which states are more preferred than others. Furthermore, the policy preferences of one person may be different than the preferences of another. For example, one person who requests tasks for a robot may prefer that it avoid traveling near people while another may prefer that it take the shortest path. Interestingly, these preferences are given in terms of features of the states and not the states themselves.

Assuming that states that can be factored into features [1], it has been shown that it is possible to define the reward in terms of these features rather than defining a reward for a whole state,action pair (*i.e.,* multiattribute utility theory [2], [3]). By defining the reward function as an additive sum of subrewards over these features, it is possible to reduce the problem of generating the reward function to one of determining how the subrewards should be scaled in relation to each other. In our previous path preference example, we can define the state as comprised of two independent state features - the x,y location and number of people present. The reward function is then defined as the sum of subrewards for distance between locations and number of people present.

While it has been common to manually tweak scaling factors of subrewards in additive reward functions until the desired policy is reached, much recent work has focused on eliciting preferences from people in order to learn them (*e.g.* [4], [5], [6], [7]). In Inverse Reinforcement Learning, an agent learns the additive reward function from demonstrated optimal behavior using linear programming [8]. In preference elicitation, users are asked to give their preference for possible states and actions in order to learn reward functions. The preferences could be direct numerical values (*e.g.,* [9], [4]) or probabilistic comparisons between multiple states and actions (*e.g.,* [6], [7]). Each of these methods has been very successful in learning additive reward functions. However, the human-computer interaction (HCI) literature says that non-expert users tend to be error-prone in estimating numerical or probabilistic preferences and they lose focus and precision when required to answer many questions or demonstrate their preferences.

In this work, we propose and evaluate a novel Monte Carlo algorithm for eliciting non-experts' preferences in additive reward functions that finds an approximate solution for the scaling factors in log of the number of Monte Carlo hypotheses generated [10]. The algorithm instantiates a number of scaling factor hypotheses and searches a finite set of example states to find a pair which most evenly divides the hypotheses. It then asks the user which of the two states she prefers with a particular action, and removes the hypotheses that are invalidated with the new preference. Based on the HCI literature, preferences for two concrete states should result in fewer errors than prior elicitation methods. The preferences form a set of inequalities that can be solved to approximate the scaling factors. Because the algorithm aims to divide the hypothesis space in half for each question, the algorithm will terminate after asking log in the number of hypotheses questions. Depending on the number questions a user is willing to answer, the number of hypotheses can be changed to optimize the scaling factors as much as possible.

The paper is organized as follows. First, we describe the related work in learning reward functions. Next, we introduce our Monte Carlo algorithm to learn scaling factors of subrewards in additive reward functions. We then evaluate our algorithm in an example robotics application. Finally, we conclude and discuss open questions and future work.

II. RELATED WORK

We are interested in elicitation techniques for additive reward functions in which the state s is decomposed into

features s_i and subreward functions $r(s_i, a)$ compute the reward of the feature for the action a. The total reward for a state s and action a is the sum of the subrewards scaled by λ_i importance factors

$$R(s,a) = \sum_i \lambda_i r_i(s_i, a) \tag{1}$$

Given subreward functions, learning the reward function requires learning the scaling factors λ_i.

Prior techniques presented below have been very successful for learning additive reward functions for intelligent agents such as robots. However, they are not feasible to deploy to non-expert users.

A. Inverse Reinforcement Learning (IRL)

The Inverse Reinforcement Learning problem learns a reward function that can explain optimal observed action choices [11]. The algorithm hinges on the fact that because the actions are optimal, the reward for taking the action a from the observed state s must be greater than any other possible action $a' \in A$ from the same state:

$$\forall a' \in A, R(s,a) > R(s,a') \tag{2}$$

This algorithm was extended to problems in which each state is factored into features and the reward function is learned over features [8]. The set of inequalities resulting from the observed behavior can be optimally solved for the reward function using linear programming.

IRL depends on optimal demonstrations to train the reward function. While experts can easily demonstrate their preferences in some problems such as robot or car navigation [8], [5], it may be difficult for novices to 1) perform optimally [12] and 2) know which demonstrations will best teach the learner [13]. Instead, novices could be given response choices about their preferences and asked to confirm them.

B. Explicit Preference Elicitation

Preference elicitation assumes that a person has a reward function that they want an agent to learn but cannot demonstrate. Instead, the learner asks the person a series of questions to elicit these preferences. Examples of preference elicitation include asking the person for a numerical reward value [9] or to confirm a numerical reward value k [4]:

$$\sum_i \lambda_i r_i(s_i, a) =? \text{ OR } \sum_i \lambda_i r_i(s_i, a) \stackrel{?}{=} k \tag{3}$$

These preferences can be used to solve for the λs even if the features s_i are not independent. However, people may not be consistent with how they assign numbers to states meaning that the elicited values may have error. Research on how to write surveys shows that the order the questions are asked influences the numerical value they are given [14] and if the questions are multiple choice the scale of the choices matters for the answers they will provide [15].

Other work has focused on asking which of two probabilistic scenarios is preferred [7]. The algorithm asks questions of the form "Would you prefer feature action a in 1) a concrete state where $s_i = s_i^\top$ and $s_{j\neq i} = s_j^\perp$ or 2) with probability p, $s = s^\perp$ and probability $(1-p)$, $s^\top$?". Here, $s^\perp$ and $s^\top$ are the absolute worst and best states and $s_j^\perp$ and $s_j^\top$ are the best and worst value of feature s_j. By setting $r_i(s_i^\top, a) = 1$, the reward for the state s is:

$$R(s,a) = \lambda_i r_i(s_i, a) + \sum_{j\neq i} \lambda_i * 0 = \lambda_i * 1 = \lambda_i \tag{4}$$

Similarly, by setting $R(s^\perp, a) = 0$ and $R(s^\top, a) = 1$, the probabilistic outcome

$$p * R(s^\perp, a) + (1-p) * R(s^\top, a) = 1 - p \tag{5}$$

Depending on the person's preference, we can constrain $\lambda_i < (1-p)$ or $\lambda_i > (1-p)$. This algorithm chooses p and i to minimize the regret of the reward function and has been shown to ask relatively few questions [7]. It is important the survey be short as response rate has been shown to decline as the number of questions increase (*e.g.,*[16]). While this approach is better than asking for values, it has been shown that people often overestimate good things to happen over the bad [17]. As a result, the values of λ may be skewed towards 0 as people are more likely to continue to choose the probabilistic choice more often. In our work, we aim to constrain λ_is by asking about two concrete states rather than probabilistic ones.

Next, we describe our algorithm to learn additive reward functions that still asks few questions but also asks questions that are less susceptible to human error.

III. Monte Carlo Learning Algorithm for Additive Reward Functions

We contribute an algorithm that asks people for their preferences between two concrete states with concrete values for each feature s_i

$$\sum_i \lambda_i * r_i(s_i = s_1, a) \stackrel{?}{>} \sum_i \lambda_i * r_i(s_i = s_2, a) \tag{6}$$

Because the states are not probabilistic and we are not asking for numerical values, related work shows that people should find this exercise to be understandable and that their preferences are consistent over the length of the survey.

The algorithm to determine the pairs of concrete states to ask about is a Monte Carlo method in which we generate a set of hypothesis λs and choose a pair that best divides the hypothesis space in half. Like the other methods of preference elicitation, dividing the space in half ensures that the algorithm asks relatively few questions (log in the number of hypotheses). The last hypothesis best approximates the person's preferred scaling factors with the error rate decreasing as the number of hypotheses increases. Additionally, because preferences form linear inequalities, it is also possible to solve the linear program to approximate λ like IRL does.

Next, we describe the algorithm in detail and present results to show the success of the algorithm.

A. Problem Definition

Concretely, let a state s be factored into features $\langle s_i \rangle$ that reflect the tradeoffs that users are making to determine their preferences (*e.g.,* time, distance, interruption, etc). Additionally, let concrete state,action pair *scenarios* with specific feature values be denoted σ where $\sigma.s$ and $\sigma.a$ refer to the scenario's state and action respectively. The additive reward function of a state,action pair $R(s,a) = \sum_i \lambda_i r_i(s_i, a)$ is the sum of scaled subrewards $r_i(s_i, a)$ computed over the state's features. Without loss of generality, we require the values of each subreward to be normalized $r_i(s_i, a) \in [0, 1]$, where values towards 1 represent the better features. The goal of our LearningImportance algorithm is to learn the values of λ_i subject to the constraints that $\forall i, 0 \leq \lambda_i \leq 1$ and $\sum_i \lambda_i = 1$.

Note that because of the subreward normalization, $\forall \lambda_i \geq 0$. Proof by contradiction. If there was a $\lambda_i < 0$, the higher the subreward the lower the product $\lambda_i * r_i(s_i, a)$. This contradicts the requirement that feature values near 1 are better. The subreward function could instead be negated (and renormalized) so that λ_i is positive. Additionally, without the constraint $\sum_i \lambda_i = 1$, there are an infinite number of valid scaling factors (*e.g.,* for numFeatures = 2, $\lambda_1 = \lambda_2 = 0.5$ is equivalent to $\lambda_1 = \lambda_2 = 1.0$)

B. Algorithm Overview

Our LearnRelationship algorithm is outlined in Algorithm 1. We instantiate the algorithm with a number of features *numFeatures* and a set of concrete state,action *scenarios* that would make sense to explain to people. Given the number of features, the algorithm generates a set of hypotheses h for the possible values of λ subject to the constraints defined above (Line 1, Figure 1a). Our algorithm also instantiates the list of user *preferences* to the constraint that $\sum_i \lambda_i = 1$.

While the number of valid hypotheses h is still greater than 1, the algorithm iterates over all pairs of *scenarios* to find the pair which divide the hypothesis space most evenly (Line 4, Figure 1b). We use a Monte Carlo method to find the best scenario pairs rather than computing the true area of the hypothesis space. When the user responds with their preference of either $(s,a)_1$ or $(s,a)_2$ (Line 5), the algorithm generates the correct preference inequality (Lines 6-10), adds that preference to the list of preferences (Line 11), and then iterates through h to remove the hypotheses that are invalidated by the new preference (Line 12) (Figure 1c). It repeats the process until it narrows down the hypothesis space to a single hypothesis (Figure 1d). Then, it finds a solution λ (Line 14).

We next discuss the details of each function in turn.

C. Generating scenarios

Because our algorithm asks about concrete state-action scenarios in a survey form, those scenarios must be explainable and understandable to people. We recommend making a list of scenarios by hand that are easy to describe to users in order to ensure that they meet this requirement. In practice, we have not found the particular scale or values of these scenarios to have an impact on the ability to accurately learn λ, except values spanning each feature be used.

Algorithm 1 LearnImportance(numFeatures, scenarios)

```
1:  h ← GenerateMonteCarloHypotheses(numFeatures)
2:  preferences ← Σ_i λ_i = 1
3:  while |h| > 1 do
4:      (σ1,σ2) ← FindBestPair(h, scenarios)
5:      betterScenario ← Ask(σ1,σ2)
6:      if betterScenario = σ1 then
7:          pref ← Σ_i λ_i(r_i(σ1.s_i, σ1.a) − r_i(σ2.s, σ2.a)) > 0
8:      else
9:          pref ← Σ_i λ_i(r_i(σ2.s_i, σ2.a) − r_i(σ1.s, σ1.a)) > 0
10:     end if
11:     preferences ← preferences ∪ pref
12:     h ← RemoveHypotheses(h, pref)
13: end while
14: return FindSolution(preferences)
```

As an example, suppose the problem is determining user preferences about how often a learning algorithm can query them, and there are two features - a user interruption level computed on a scale $[1, 10]$ and the number of hours ago that the last query was asked $[0, 24]$. While a valid scenario is any combination of these two features, it may not be easy for a user to imagine a situation where their interruptibility was 5.346 and they were asked 8.82 hours ago. Instead, we could create concrete scenarios that require the interruptibility be given in whole numbers and last question asked hours ago could be [0.5, 1, 2, 4, 8, 24] (hour values that are relatively easy to think about). The generated scenarios should be easy for a user to understand without much explanation.

D. GenerateMonteCarloHypotheses

We generate a uniform random set of hypotheses for λ subject to our constraints that $0 \leq \lambda_i \leq 1$ and $\sum_i \lambda_i = 1$. Each hypothesis is in $\mathbb{R}^{|\lambda|-1}$, as λ_0 can be completely explained by the remaining features. As with all Monte Carlo methods, increasing the number (and therefore the density) of hypotheses increases the accuracy of the learned λ. In our experiments section, we vary the number of generated hypotheses to show how the accuracy of λ increases.

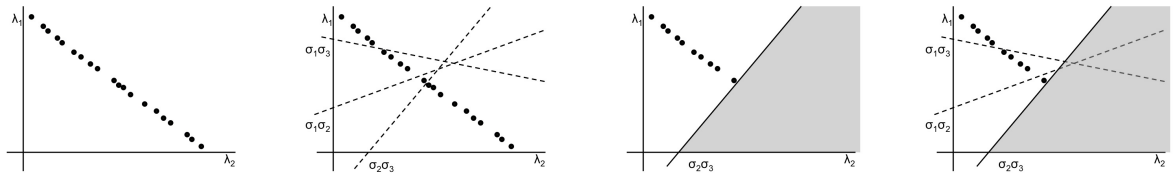
E. FindBestPair

The additive reward for each scenario is $R(s,a) = \sum_i \lambda_i r_i(s_i, a)$. If a user preferred scenario σ_1 over σ_2, we can generate an inequality

$$\sum_i \lambda_i r_i(\sigma_1.s, \sigma_1.a) > \sum_i \lambda_i r_i(\sigma_2.s, \sigma_2.a)$$
$$\text{or } \sum_i \lambda_i (r_i(\sigma_1.s, \sigma_1.a) - r_i(\sigma_2.s, \sigma_2.a_2)) > 0 \qquad (7)$$

which forms a plane through the hypothesis space h and invalidates hypotheses that do not satisfy the new preference.

Let $invalid(h,\text{pref})$ be the number of hypotheses in h rendered invalid by the preference. FindBestPair iterates



(a) (b) (c) (d)

Fig. 1. a) The LearnImportance algorithm first GeneratesMonteCarloHypotheses for λ where $\sum_i \lambda_i = 1$ (here $|\lambda| = 2$). b) The algorithm searches the through all pairs of scenarios to FindBestPair that cuts the hypothesis space in half. c) When it finds the best pair, it asks the person for their preference and removes the invalidated hypotheses. d) Then, it repeats the process of finding a new pair of scenarios, asking, and removing the invalid hypotheses.

through all pairs and finds scenarios σ_1 and σ_2 that minimize

$$abs(invalid(h, \text{pref}) - |h|/2) \tag{8}$$

where pref is an equality like Equation 7. In other words, it finds the pair of scenarios that best splits the hypotheses in half. Dividing h in half each time through the loop (Line 3) implies that the number of user preference questions needed to learn λ is $log_2(|h|)$ questions.

F. RemoveHypotheses

Once the best pair is found, the user is asked for their preference, listing the features of each scenario. Then the algorithm generates the inequality preference *pref* based on their response. The RemoveHypotheses function iterates through all hypotheses in h, determining whether each is valid by setting in the hypothesized values of λ into the inequality, and removing those from the list that are invalid. The hypotheses remaining in h after RemoveHypotheses satisfy all preference inequalities in *preferences*.

Alternatively to removing hypotheses, it is possible to redistribute the invalid hypotheses in the valid area keeping the number of hypotheses the same over time. This process will increase the accuracy of the algorithm because there are more hypotheses to divide in half at each step, but at the cost of possibly asking the person more questions.

G. FindSolution

The FindSolution algorithm could use the remaining hypothesis as its approximation of λ. As the number and density of hypotheses increases, the error of the remaining hypothesis decreases because the valid hypothesis area is more constrained [10].

It is also possible to evaluate the linear program of the preference inequalities that were generated through the algorithm to constrain the hypothesis space. An algorithm such as the simplex algorithm [18] find the valid area within the inequalities and and find the minimum or maximum point subject to another constraint (*e.g.,* minimize or maximize the distance from (0,0)) To understand how the selected scenarios impacted the preference inequalities, we will evaluate the linear program in our following experiments.

Next, we present experiments towards demonstrating the use of our algorithm on an example additive reward problem.

IV. Experiments

In order to understand how our algorithm performs on real-world tasks, we present our experiments to evaluate its use on an example robot application with an additive reward function. We first describe our application and the personalized reward functions that we would like to learn. Then, we describe our surveys of real people to understand the usability of the scenario comparison questions for our application. Finally, we present our simulated results to learn random scaling factors λ with different numbers of generated hypotheses and different types of generated scenarios for $|\lambda| = 2$ and 3 (number of constraints in the example).

A. Example Application: Robots that Request Human Help

Our interest in learning additive reward functions stems from our work human-robot interaction. We focus on robots that perform tasks in human environments and request help from humans when necessary (*e.g.,* to push elevator buttons, pour coffee, or perform other manipulation tasks as our robot has no manipulators). Its state includes information about its own location as well as information about offices and people in the environment. In order to determine where to navigate in order to ask for help, our robot determines the offices that are closest to its current location and the location where it will need help (*e.g.,* the elevator, kitchen, etc). Additionally, because the robot is in our human environment long-term, we realize that it must also take into account a "cost" of asking each person for help, in terms of how interruptible they are, how frequently they help, and the last time they were asked. As a result, our robot minimizes the sum of three features computed over the state to determine where to navigate and who to ask for help:

$$\arg\min_{l_o} \texttt{COT}(l_{curr}, l_o) + \texttt{COA}(l_o, \iota_o, f_o, t_o) + \texttt{COTH}(l_o, l_{help})$$

- $\texttt{COT}(l_{curr},l_o)$: distance to travel from current location l_{curr} to office l_o,
- $\texttt{COA}(l_o,\iota_o,f_o,t_o)$: cost of asking at l_o for help, based on their interruptibility ι_o, frequency of help f_o, and the last time t_o they helped,
- $\texttt{COTH}(l_o,l_{help})$: distance to travel from the office l_o to the location of help l_{help} with the human.

In our problem, our three features: cost of asking (COA) and the cost of traveling with and without the human (COT and COTH) are independent and additive. While we could

manually tweak subreward functions so that the values are roughly in the range to get to our desired behavior, this takes time and is subject to error. Because each person has different preferences, determining the COA and COTH for each office is infeasible without asking them for their preferences but is also infeasible to require them to demonstrate their preferences. We would like the people in our building to fill out a survey of their own preferences for when they get asked for help in order to increase their satisfaction with the robot and likelihood that they will continue helping it over time. The survey should not be too long in order to increase the likelihood that it gets filled out [16], and should include questions that are easy to answer and not susceptible to error.

Using our algorithm, we will train 3 scaling factors for our three subreward problems as a function of our features:

$$\lambda_1 * r_1(COT, ask(l_o)) + \lambda_2 * r_2(COA, ask(l_o)) + \lambda_3 * r_3(COTH, ask(l_o)) \quad (9)$$

Interestingly, in this problem, we also have a secondary additive reward learning problem in relationship between our 3 COA features - l_o,ι_o,f_o, and t_o - to compute the COA for the larger reward function:

$$\lambda_4 * r_4(\iota_o, ask) + \lambda_5 * r_5(f_o, ask) + \lambda_6 * r_6(t_o, ask) \quad (10)$$

We took a two part approach in order to learn these scaling factors. First, we deployed a survey with 50 of these comparison questions in order to evaluate the consistency of responses and the ease of understanding from participants. These questions were not dynamically generated using our algorithm; instead they were static and the same for each person. Then, we performed simulated experiments to understand how our algorithm performed with different preferences, scenarios, and numbers of hypotheses.

B. Survey Results

We conducted a web survey about participants preferences for what conditions and how frequently they would be willing to help our robot. In the first half of the survey, subjects were shown a partial map of our building with different configurations of people in offices who could be available, different locations of the robot, and different locations for receiving help - the elevator or the kitchen to make coffee. They were asked which person the robot should choose to ask for help (Figure 2). In the second half of the survey, participants were told that they were the ones being asked for help and answered questions about their willingness to help the robot under different conditions of interruptibility, recency of the last time the robot could have asked for help, and frequency of the number of questions the robot could ask per week. Thirty participants were recruited through a local recruiting website for human-subject studies. The survey contained 50 questions and took about 45 minutes for participants to complete.

Participants were able to successfully answer all of our concrete scenario comparison questions and did not ask us to clarify the scenarios or the instructions. As expected, they did respond that the 50 questions was extreme for an online survey format. We analyzed the results to understand if and when participants' responses began conflicting with previous answers. Importantly, while participants did tire of the questions and become careless in their responses (sometimes conflicting with previous answers), the first 10-20 questions depending on the person were consistent. This maximum number of questions is in line with related work in survey responses and indicates that our questions are not more susceptible to human error as the questions asking about probabilities.

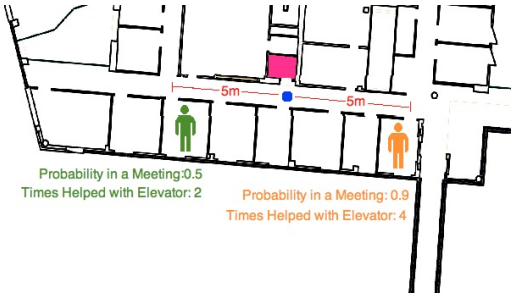


Fig. 2. In the survey, participants were asked which of two people (orange and green) should help a robot (center in blue) use the elevator (pink box). For example, the people here are equidistant from the elevator but have different availability and frequency of helping.

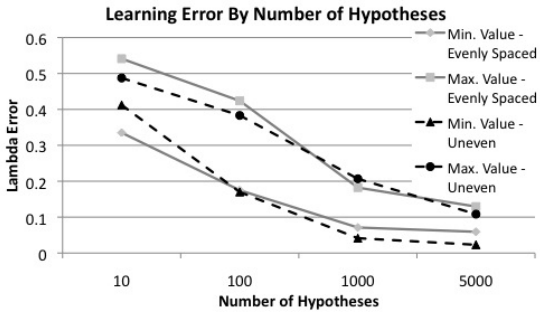


Fig. 3. As the number of hypotheses increases for $|\lambda| = 3$, the error rate of the solutions decreases regardless of the evenness of the scenarios chosen.

C. Simulated Results

In order to understand how our algorithm performed with different scenarios and different numbers of generated hypotheses for random scaling factor preferences, we performed a series of simulated experiments. In the experiments, we randomly generated sets of true λ such that $\sum_i \lambda_i = 1$, and then ran our algorithm with auto-generated responses to the Ask questions based on the true λ. In total, we tested 30 random λs for each condition, varying $|\lambda| = 2$ or 3, the number of hypotheses (number of queries), preference solution analysis, and types of scenarios generated.

First, while it is possible to use the remaining hypothesis as the solution for our algorithm, we analyzed the linear inequality preferences to find the minimum and maximum distance solution from (0,0) based on these constraints using the simplex algorithm. Notice that these solutions may not satisfy the $\sum_i \lambda_i = 1$ requirement. Figure 3 shows the average minimum and maximum distance error of the solution λ from the randomly generated true λ for $|\lambda| = 3$.

The error rates of both solutions are parallel and decreasing with the number of hypotheses indicating that the preference constraints are narrowing down λ from the high and low values. In our scenarios, the minimum distance solution from (0,0) had less error than the maximum distance solution with average difference 0.14.

Next, we evaluate our choices in scenarios and the resulting accuracy. We generated both evenly spaced scenarios for our robot navigation example (whole number evenly spaced distances, values of a potential cost of asking, and cost of traveling with the human) as well as unevenly spaced scenarios (interruptibility in tenths, hours ago of last question [0.5, 1, 2, 4, 8, 24, 48, 168], and frequency of questions per day [0.05, 0.1, 0.5, 1.0, 2.0, 5.0]). Our subreward functions map these value to $[0, 1]$ in similarly uneven spacing. Because these scenarios dictate the dividing hyperplanes through the hypothesis space, it is possible that they could affect the error of the learned λ. The grey lines in Figure 3 represent the evenly spaced conditions and the black dashed lines represent the uneven ones. The differences between the lines is negligible compared to the number of generated hypotheses, indicating that the algorithm is accurate irrespective of the scenarios.

Finally, we compare the number of hypotheses or questions generated compared to the accuracy of the solution based on the size of λ. As Figure 3 shows, as the number of hypotheses and number of questions = log(hypotheses) increases, the error decreases. With 5000 hypotheses or 12 questions, the algorithm is able to learn the solution with 0.02 (s.d. 0.1) error on average. The high standard deviation implies that most of the solutions had 0.0 error. For $|\lambda| = 2$, the algorithm required only 5 questions on average to reach error 0.015(s.d. 0.05) although increasing the number of hypotheses to 5000 reduced error to 0.0(s.d.0.001). Rather than adding 10 times as many hypotheses for each $|\lambda|$ increase, it is possible to redistribute the hypotheses in the valid area rather than removing them as we mentioned above. This would reduce the computation of iterating through all hypotheses to find the pair that best pair of scenarios that divides them without sacrificing accuracy.

V. Conclusion

Many robots compute reward functions over their states and actions to determine their optimal policy. We focused on learning additive reward functions for states that can be factored into features. As an example application, we described our human-robot interaction interests in which a robot determines who in the environment to ask for help when it cannot perform an action. This application requires learning the importance relationships between navigation time, human interruption and travel cost features.

While there have been many solutions to solving this problem, we showed that they can be error-prone when novices train them. We contribute a novel Monte Carlo algorithm for eliciting preferences from people including novices and learn additive reward functions that learns solutions in log of the number of generated Monte Carlo hypotheses. In particular, our algorithm asks people for their preferences between two concrete state,action scenarios and refines its list of valid hypotheses accordingly. We demonstrate using a survey that real people can understand the questions and answer accurately. Then, we provided simulated results to show that the error decreases as the number of hypotheses increases and that the specific concrete scenarios that the algorithm asks about do not affect the accuracy. We conclude that our algorithm is less susceptible to human error than prior work with learning time proportionate to the number of hypotheses.

Acknowledgments

This work was partially supported by National Science Foundation award number NSF IIS-1012733 and a National Physical Science Consortium Fellowship. The views and conclusions contained in this document are those of the author and should not be interpreted as representing the official policies, either expressed or implied, of any sponsoring institution, the U.S. government or any other entity.

References

[1] C. Boutilier, T. Dean, and S. Hanks, "Decision theoretic planning: Structural assumptions and computational leverage," *JAIR*, vol. 11, pp. 1–94, 1999.

[2] P. C. Fishburn, "Interdependence and additivity in multivariate, unidimensional expected utility theory," *Int. Econ. Rev.*, vol. 8, pp. 335–342, 1967.

[3] R. L. Keeney and H. Raiffa, *Decisions with Multiple Objectives: Preferences and Value Trade-offs*. Wiley, 1976.

[4] D. Braziunas and C. Boutilier, "Preference elicitation and generalized additive utility," in *Twenty-First Conference on Artificial Intelligence (AAAI-06)*, 2006.

[5] B. D. Ziebart, A. Maas, J. A. Bagnell, and A. K. Dey, "Maximum entropy inverse reinforcement learning," in *23rd National Conference on Artificial Intelligence*, 2008, pp. 1433–1438.

[6] G. Hines, "A study in preference elicitation under uncertainty," University of Waterloo, Tech. Rep., 2011.

[7] K. Regan and C. Boutilier, "Eliciting additive reward functions for markov decision processes," *Twenty-Second Joint Conference on Artificial Intelligence (IJCAI 2011)*, 2011.

[8] P. Abbeel and A. Y. Ng, "Apprenticeship learning via inverse reinforcement learning," in *ICML*, 2004, pp. 1–8.

[9] C. Gonzales and P. Perny, "Gai networks for utility elicitation," in *9th Intl. Conference on Principles of Knowledge Representation and Reasoning (KR-04)*, 2004, pp. 224–234.

[10] N. Metropolis and S. Ulam, "The monte carlo method." *Journal of the American Statistical Association*, vol. 44, p. 335, 1949.

[11] A. Y. Ng and S. Russell, "Algorithms for inverse reinforcement learning," in *ICML*, 2000, pp. 663–670.

[12] B.Argall, B.Browning, and M.Veloso, "Learning robot motion control with demonstration and advice-operators," in *IROS 08*, 2008.

[13] M. Eagle and E. Leiter, "Recall and recognition in intentional and incidental learning," *Journal of experimental psychology*, vol. 68, pp. 58–63, 1964.

[14] N. Schwarz and H. J. Hippler, "Subsequent questions may influence answers to preceding questions in mail surveys," *Public Opinion Quarterly*, vol. 59, no. 1, pp. 93–97, 1995.

[15] N. Schwarz, B. Knauper, H. J. Hipler, E. Noelle-Neumann, and L. Clark, "Numeric values may change the meaning of scale labels," *Public Opinion Quarterly*, vol. 55, no. 4, 1991.

[16] T. A. Heberlein and R. Baumgartner, "Factors affecting response rates to mailed surveys: A quantitative analysis of the published literature." *American Sociological Review*, vol. 43, 1978.

[17] W. Weiten, *Psychology: Themes and Variations*. Wadsworth Publishing, 2011.

[18] G. B. Dantzig, *Linear Programming and Extensions*. Princeton University Press, 1963.