Multiagent Environment Design in Human Computation (Extended Abstract)

Chien-Ju Ho
Computer Science Department
University of California, Los Angeles
cjho@cs.ucla.edu

Yen-Ling Kuo and Jane Yung-jen Hsu
Computer Science & Information Engineering
National Taiwan University
{b94029,yjhsu}@csie.ntu.edu.tw

Categories and Subject Descriptors

I.2.11 [**Distributed Artificial Intelligence**]: Multiagent systems

General Terms

Algorithms, Design

Keywords

Human Computation, Environment Design, Collaborative Filtering

1. INTRODUCTION

Human computation aims to solve computationally-hard problems, e.g. image tagging or commonsense collection, by utilizing collective human brain power. There are a variety of applications available nowadays. Games with A Purpose (GWAP) [4] engage players in an online game and let them help solve tasks while having fun. Crowdsourcing markets, such as Amazon Mechanical Turk (http://mturk.com), provide platforms for workers to contribute their brain power in exchange for monetary rewards. Peer productions systems, e.g. Wikipedia or Yahoo! Answers, let online users construct knowledge bases for common good.

Despite the impressive progress of developing applications to solve real-world problems, little study is conducted in theory to guide the design of human computation systems. von Ahn and Dabbish [4] discussed the design principles of Games with A Purpose. Some other researchers [3] analyzed the incentive structure of human computation systems in a game theoretic approach. While these projects addressed the design of the system mechanisms, many situations arise when the developers do not have full privilege to modify the systems. For example, developers on Mechanical Turk cannot change the way they interact with the workers. They can only make little modifications, such as the size of payments, or the task descriptions, to encourage workers complete the tasks quickly and accurately. In this work, we focus on situations in which developers can only make limited changes to the systems. In particular, we view this problem as an environment design problem with multiple agents.

Cite as: Multiagent Environment Design in Human Computation (Extended Abstract), Chien-Ju Ho, Yen-Ling Kuo and Jane Yung-jen Hsu, *Proc. of 10th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2011)*, Tumer, Yolum, Sonenberg and Stone (eds.), May, 2–6, 2011, Taipei, Taiwan, pp. XXX-XXX.

The concept of environment design was first proposed by Zhang and Parkes [6], where they considered the interested party tries to influence agent's behaviors in a Markov Decision Process (MDP) setting, and the interested party can only make limited changes to the reward function. In their work, they proposed solutions to find the optimal environment for a single agent by active indirect elicitation algorithms. The agent's private information is inferred by observing its responses to incentive changes. They then extended their work to a more general framework [5] and provided algorithms for problem solution. Inspired by their work, Huang et al. [2] conducted an empirical study applying environment design framework for automatic task design in Amazon Mechanical Turk.

In this work, we incorporate collective information from multiple agents to enhance the environment design framework. We extend the environment design problem to settings with multiple agents in the context of human computation. While users usually perform independent tasks in human computation systems, we assume there is no agent interactions. Another assumption is that users will take actions similar to the actions taken by like-minded users or users with similar abilities. We propose two approaches to utilize collective information. First, we assume the existence of the agent types and propose an algorithm for agent type elicitation. We then relax the assumption of agent types by adopting the collaborative filtering technique.

2. MULTIAGENT ENVIRONMENT DESIGN

Our model is an extension of Zhang and Parkes's work [5]. The only change is the introduction of multiple agents. A multiagent environment design problem in human computation consists of agents, environments, agent model parameters, agent decision space, agent decision function, and a utility function. The goal of the problem is to maximize the total utility value by finding the best environment for each agent using the histories of the agent's behaviors.

Take developers on Mechanical Turk for example, agents are workers, environments are possible modifications developers can make, agent decision space is the set of actions workers can take, agent model parameters are the private information of workers, and the utility function describes how desirable the workers' actions are to the developers.

Clearly, without any assumption about the agents, we cannot do any better than the one-agent environment design problem. The fundamental assumption in this work is that agents will take actions similar to the actions that like-minded agents or agents with similar abilities take.

2.1 Agent type elicitation

We first assume that agents fall into a relatively small set of "types". Agents of the same type will take the same actions in all environments. In the following discussion, the number of agents is denoted as m, and the number of agent types is denoted as k. If the types of agents are known a priori, the agent behaviors of the same type can be used together to elicit agent model parameters. Therefore, the convergence speed is trivially $O(m/k)$ times faster than the one-agent algorithm proposed by Zhang et al.[5] However, it is usually not possible to know the types of agents in advance. Therefore, we propose an algorithm, which picks an environment set $\mathcal{E}$ as pre-testing rounds, to elicit agent types. The agent type information is then used to help speed up the elicitation of model parameters.

It is clear that the performance of the algorithm highly depends on the choice of the environment set $\mathcal{E}$. In the following definition, we provide a measure for how good the environment set can be used to distinguish agent types.

DEFINITION 1. *(p-separable) The agent types are called p-separable in environment set $\mathcal{E}$ if for any two agents of different types, the fraction of the environments set $\mathcal{E}$ that they choose different actions is larger than p.*

The smaller value of p means that agents of different types are less likely to take the same actions in the environment set $\mathcal{E}$. Therefore it is easier to distinguish different types of agents in the environment set $\mathcal{E}$. In real-world applications, the developers usually have prior knowledge about the environments, e.g. task types in Mechanical Turk. Therefore, they could choose representative environments (i.e. minimizing p) to speed up the convergence of classifying agents.

LEMMA 1. *If the agent types are* p-separable *in environment set $\mathcal{E}$ and $|\mathcal{E}| = r$, the probability of eliciting the wrong agent type after observing r environments would be less than $(k-1)p^r$.*

Assume there are 10 types of agents ($k = 10$), and the environment set $\mathcal{E}$, where $|\mathcal{E}| = 10$, is 0.5-separable for agents. Then the probability of wrongly classifying the agents are less than 1%.

2.2 Collaborative filtering

In this section, we relax the agent type assumption and propose a collaborative filtering approach. If we record the utility values of agent's past actions in a matrix, the problem we are solving is to find the environment with highest utility value for each agent. This is actually an *environment recommendation* problem in which developers aim to find the best environment for each user to maximize the utility value. Since this problem is in a standard format of collaborative filtering. Any collaborative filtering algorithms can be applied. In this work, we are more curious about if we can design an algorithm to achieve high accuracy of recommendations with few matrix entries.

Given the assumption that agents take actions similar to the actions taken by like-minded agents, it is implied that the matrix has a good low rank approximation. In the following discussion, we first talk about the result of Drineas et al. [1] and then interpret how their result can be applied in our problem settings. In their algorithm, they random sample c columns and r rows of the matrix A. They can then provide a prediction matrix $\hat{A}$, where the expected error between A and $\hat{A}$ satisfies the following lemma.

LEMMA 2. *([1]) Let $\sigma_t, t = 1, \ldots, \rho$ denote the singular value of A. Then, the algorithm shows the error satisfies*

$$E(\|A - \hat{A}\|_F^2) \leq \sum_{t=k+1}^{\rho} \sigma_t^2 + (\sqrt{\frac{k}{c}} + \frac{k}{r})\|A\|_F^2$$

In the lemma, picking $\mathcal{O}(k/\epsilon)$ rows and $\mathcal{O}(k/\epsilon^2)$ column bounds the expectation of relative error to $\lambda + \epsilon$, where λ is the relative error between the actual and low-rank matrix. Therefore, if we can find volunteers to test all environments (r rows) and ask the other users to perform test rounds (c columns), Lemma 2 can be used to provide an error bound. Though it is often infeasible to ask a small number of agents to test all the environments, we could still get some intuitions about how good the approximation could be. Developing new algorithms to avoid "sampling all environment" is an interesting future research direction.

3. CONCLUSION

In this work, we extend the environment design problem to settings with multiple agents in the context of human computation. To incorporate the collective information from multiple agents, we propose two approaches, agent type elicitation and collaborative filtering, under different assumptions. The formulation and algorithms provide solutions for developers in human computation systems to find the environment settings maximizing their goal functions. Future work should continue to explore the aspects of agent interactions, integrations to the mechanism design of human computation, and applying the results to real-world applications.

Acknowledgements

We would like to thank Jennifer Wortman Vaughan for the helpful comments for this paper. This research was supported in part by grant #NSC 99-2221-E-002-139-MY3 from the National Science Council of Taiwan.

4. REFERENCES

[1] P. Drineas, I. Kerenidis, and P. Raghavan. Competitive recommendation systems. In *Proceedings of STOC'02*, pages 82–90, New York, NY, USA, 2002. ACM.

[2] E. Huang, H. Zhang, D. C. Parkes, K. Z. Gajos, and Y. Chen. Toward automatic task design: a progress report. In *HCOMP'10*, pages 77–85, New York, NY, USA, 2010. ACM.

[3] S. Jain and D. C. Parkes. A game-theoretic analysis of games with a purpose. In *WINE 2008*, pages 342–350, Dec. 2008.

[4] L. von Ahn and L. Dabbish. Designing games with a purpose. *Communications of the ACM*, 51(8):58–67, 2008.

[5] H. Zhang, Y. Chen, and D. C. Parkes. A general approach to environment design with one agent. In *Proceedings of IJCAI'09*, Pasadena, CA, USA, 2009. ACM.

[6] H. Zhang and D. C. Parkes. Value-based policy teaching with active indirect elicitation. In *Proceedings of AAAI'08*, 2008.