HARVARD UNIVERSITY
Graduate School of Arts and Sciences

DISSERTATION ACCEPTANCE CERTIFICATE

The undersigned, appointed by the

School of Engineering and Applied Sciences

have examined a dissertation entitled

"Computational Environment Design"

presented by Haoqi Zhang

candidate for the degree of Doctor of Philosophy and hereby certify that it is worthy of acceptance.

Signature ______________________

Typed name: Prof. D. Parkes

Signature ______________________

Typed name: Prof. Y. Chen

Signature ______________________

Typed name: Prof. K. Gajos

Signature ______________________

Typed name: Prof. E. Horvitz (Microsoft)

Signature Robert C. Miller

Typed name: Prof. R. Miller (MIT)

Date September 5, 2012

Computational Environment Design

A dissertation presented
by

Haoqi Zhang

to
The School of Engineering and Applied Sciences
in partial fulfillment of the requirements
for the degree of
Doctor of Philosophy
in the subject of

Computer Science

Harvard University
Cambridge, Massachusetts
September 2012

Thesis advisor

David C. Parkes

Author

Haoqi Zhang

Computational Environment Design

Abstract

The Internet has evolved into a platform on which large numbers of individuals take action and join in collaborations via crowdsourcing, social media, and electronic commerce. When designing social and economic systems on the Internet, a key challenge is understanding how to promote particular desired behaviors and outcomes. I call this problem *computational environment design*.

Notable abilities afforded by the Internet, such as the ability to recruit large numbers of individuals to join problem-solving efforts via crowdsourcing and social media, and the ability to engage in a data-driven iterative design process, are creating new opportunities and inspiring new methods for computational environment design. This dissertation focuses on these abilities and proposes an approach for arriving at effective designs by *reasoning and learning about characteristics of participants and how these characteristics interact with a system's design to influence behavior*.

The dissertation consists of two major components. The first component focuses on designing *crowdsourcing* and *human computation* systems that leverage a crowd to solve complex problems that require effective coordination among participants or the recruitment of individuals with relevant expertise. I show how reasoning about crowd abilities and limitations can lead to designs that make crowdsourcing *complex tasks* feasible, effective, and efficient. The solutions introduce new design patterns and methods for human computation and crowdsourcing; notable contributions include a *crowdware* design for tackling human computation tasks with global constraints, and incentive mechanisms for *task routing* that harness people's expertise and social expertise by engaging them in both problem solving and routing.

The second component focuses on understanding how to design effective environments automatically. I introduce a general *active, indirect elicitation* framework for *automated environment design* that learns relevant characteristics of participants based on observations of their behavior and optimizes designs based on learned models. Theoretical contributions include developing an active, indirect elicitation algorithm for a sequential decision-making setting that is guaranteed to discover effective designs after few interactions. Practical contributions include applications of the active, indirect elicitation framework to crowdsourcing. Specifically, I demonstrate how to automatically design tasks and synthesize workflows when optimizing for desired objectives given resource constraints.

Contents

Title Page . . . i
Abstract . . . iii
Table of Contents . . . v
Acknowledgments . . . vii
Dedication . . . ix

1 Introduction 1
1.1 Crowdsourcing Complex Tasks . . . 5
1.2 Automated Environment Design . . . 11
1.3 Limitations . . . 12
1.4 Thesis and Contributions . . . 13
1.5 Thesis Overview . . . 15
1.6 For the Reader . . . 18
1.7 Bibliographic Notes . . . 18

2 Human Computation Algorithms 20
2.1 Design Patterns . . . 21
2.2 Case Study: Audio Transcription . . . 26
2.3 Case Study: Nutrition Analysis . . . 38
2.4 Discussion . . . 57

3 Human Computation with Global Constraints 59
3.1 Related Work . . . 63
3.2 Mobi: A System for Crowd Itinerary Planning . . . 65
3.3 Experiment: Todo or Not Todo . . . 71
3.4 End-to-End User Study . . . 79
3.5 Discussion . . . 84
3.6 Summary and Research Directions . . . 86

4 Harnessing Crowd Abilities: Control and Synthesis 88
4.1 Related Work . . . 90

4.2 Crowd as Controllers 91
4.3 Towards Human Program Synthesis 96
4.4 Discussion 108

5 Task Routing 111
5.1 Related Work 114
5.2 Task Routing for Prediction Tasks 115
5.3 Routing Scoring Rules 118
5.4 Common Knowledge 122
5.5 Local Common Knowledge 126
5.6 Simulations and Results 136
5.7 Discussion 139

6 Automated Environment Design 142
6.1 Model for Automated Environment Design 145
6.2 An Active, Indirect Elicitation Framework 151
6.3 Case Study: Policy Teaching 153
6.4 Discussion 168

7 Automated Task Design 172
7.1 Related Work 175
7.2 Automated Task Design on Mechanical Turk 177
7.3 The Image Labeling Task 180
7.4 Measuring Output Variability 183
7.5 Initial Experiments and Behavioral Models 185
7.6 Design Experiment 194
7.7 Discussion 199

8 Automated Workflow Synthesis 202
8.1 Related Work 205
8.2 Automated Workflow Synthesis 208
8.3 An Active, Indirect Elicitation Approach 210
8.4 Human Sorting Tasks 215
8.5 Experiments 224
8.6 Discussion 233

9 Conclusion 236
9.1 Brief Review 242
9.2 Research Directions 246
9.3 One More Thing 251

Bibliography 252

Acknowledgments

David, thank you. You are the most wonderful advisor a graduate student can hope to have. You helped me grow into me, always guiding but never constraining. As I have told you before, I worry to no end that I won't be as good an advisor as you have been to me. I am going to keep trying though – you taught me that, too.

To my entire committee, thank you for the help, support, advice, everything. I don't deserve all this and can only pay you back by passing it on.

Yiling, thanks for convincing me to focus on research. I really like it now! Thanks for all the long meetings and not watching the clock when you probably should have.

Krzysztof, I am so glad you came to Harvard. We work well together and never fail to get a paper accepted. Can we keep the streak alive? I hope we try.

Eric, working with you at MSR and thereafter has been the best. You broaden my perspective. I love our conversations. Thanks for everything.

Rob, thanks for showing me those HCI ropes, being an amazing collaborator, and welcoming me into your group with open arms.

Thank you all again.

Giro, Sven, and John – I will miss your interesting thoughts and perspectives keeping me up at night in conference hotels, and me doing the same to you. To all the wonderful people in EconCS and AIRG, thank you. Special thanks to Friday afternoon tea and cakes. Keep baking it yourself, folks.

Thanks to Greg Little for getting me interested in crowdsourcing, Lydia Chilton for helping me set up and get right, and Edith Law for taking it with me to a whole other dimension. Thanks to folks in UID for providing helpful feedback, and generally being wonderful people.

Thanks to Aviv Zohar and Ariel Procaccia. You guys are awesome to collaborate with. Thanks to the wonderful undergrads I had the opportunity of working with – Eric Huang, Jerry Kung, Dylan Lake, Beatrice Liem, Eric Hysen, and Jon Noronha.

I am grateful for the generous support from the NSF and NDSEG fellowships. And to Harvard, we did it. You finally got rid of me before I reached double digits.

Shout-out to my boys from NYC – BK, Vin, Ed, Ming. We can be so stupid at times, but hey.

Mom and dad, I love you so much. To all my relatives in Beijing and around the world, thanks for always believing in me.

Laura, I love you forever. You are all that is good in me. Let's get our Stinky and grow old together.

To family, friends, and people.

- Haoqi

Chapter 1

Introduction

The Internet has evolved into a platform on which large numbers of individuals take action and join in collaborations. Just a decade or two ago, the Internet was used primarily as a source of information. Today, the Internet is a center for social and economic activity. In social computing systems such as Wikipedia and on crowdsourcing platforms such as Amazon Mechanical Turk, large numbers of individuals contribute to problem-solving efforts as volunteers and as paid workers. On social media services such as Facebook and Twitter, friends and followers communicate news, share thoughts and ideas, and engage in social and political action. In electronic markets such as eBay, Etsy, and Amazon, consumers make purchases and help one another with purchasing decisions by contributing ratings and reviews. Over the years more and more activity is taking place online, and this trend promises to continue as the Internet continues to evolve.

In hosting platforms and services, the Internet is a virtual space on which designers build systems in which participants take action. From the designer's perspective, the goal of a system is to attract participants and promote particular desired actions and outcomes. For example, Wikipedia seeks to attract contributors to write, review, and edit articles. Requesters on Mechanical Turk want to recruit workers to complete tasks well and on-time. Facebook and Twitter want users to contribute content, communicate with other users, and generally make use of available features.

I refer to the problem of designing social and economic systems on the Internet to promote desired actions and outcomes as *computational environment design.* The designer's role is to construct the *decision environment* in which participants take action. The decision environment may include interfaces, workflows, feedback to users, incentives, constraints on actions, rules and policies, and so forth. Participants have their own knowledge and abilities, interests and motivations, availability, and decision-making processes. Together with the decision environment, all of these elements influence participants' decisions about what actions to take in a system.

To elicit desired behaviors, a designer must construct a decision environment with which to drive participant actions. For example, an effective decision environment may include tools that enable collaboration among contributors, monetary and social rewards for taking desired actions, or interface elements that display relevant information for decision-making. Depending on the decision environment being constructed, effective designs may draw on principles and methods from fields such as human-computer interaction, artificial intelligence, decision science, psychology, sociology, and economics.

Notable abilities afforded by the Internet are creating new opportunities and inspiring new methods for computational environment design. One example is the ability to recruit a crowd of individuals to join in problem solving and discussion. By employing workers in an online labor market such as Amazon Mechanical Turk and reaching out to friends and followers through social media services such as Facebook and Twitter, one can now draw on a crowd to contribute to seemingly arbitrary tasks of interest. Taking advantage of this ability, *crowdsourcing* and *human computation* systems are attracting large numbers of participants to solve large-scale problems. While individuals in the crowd may only be involved briefly, and while any given individual may provide noisy inputs, we are beginning to develop mechanisms for coordination and quality control that enable a crowd to provide useful solutions in a variety of settings.

One can envision a future in which the *distributed intelligence* of humans and machines across networks are brought together to tackle complex problems, with streams of tasks flowing seamlessly to the people who are most willing and able to contribute. Despite individual limitations, crowds of humans and machines may be able to perform complex tasks that cannot be solved by humans or machines working independently. A key challenge in realizing this vision is understanding how to design decision environments that help to recruit individuals with relevant expertise to join a problem-solving effort, and that enable effective coordination and collaboration.

Another notable ability afforded by the Internet with implications for computational environment design is the ability to engage in a data-driven, iterative design process. Designers can experiment with alternative designs by modifying some aspects of a system, and track complex individual and group behaviors across multiple interactions to get rapid feedback on designs. For example, tools for A/B testing are enabling designers to iteratively make changes to their web services to better promote desired outcomes by putting hypotheses to the test and objectively measuring the outcomes of competing designs.

The ability to track complex behaviors and redesign easily not only allows designers to compare alternative designs, but also allows them to gain new insights into participants' motivations and decision-making processes. Currently, the iterative design process is largely manual and ad hoc. Designers come up with alternative designs themselves, and the experimentation process is aimed at "hill-climbing" to a local maximum. This process is not only tedious for the designer, but may miss out on parts of the design space where better solutions exist. As improved models of participant behavior and computational tools for understanding participants from data become available, one can envision a future in which we can design decision environments *automatically* by systematically discovering interventions tailored to the preferences and capabilities of participants. Such methods aim to provide for more efficient iterative design processes, that seamlessly combine domain knowledge with machine-driven processes that refine models based on observed behavior.

In this dissertation, I introduce principles and methods for crowdsourcing complex tasks and for automated environment design. I demonstrate how to discover solutions to computational environment design problems manually and automatically. A common thread in my approach is to construct effective designs by *reasoning and learning about characteristics of participants and how these characteristics interact with the decision environment to influence behavior.* By reasoning, I mean the action of thinking about participants and a design problem using available knowledge. By learning, I mean the acquisition of knowledge about participants through experience or study that informs design decisions.

In the first part of the dissertation, I show how reasoning about crowd abilities and limitations can lead to designs that make crowdsourcing complex tasks feasible, effective, and efficient. I demonstrate a number of design patterns that allow the crowd to effectively coordinate and contribute to complex tasks, and provide incentive mechanisms that encourage individuals to both contribute to a task and route the task to others who can further contribute.

In the second part of the dissertation, I provide a general framework for *automated environment design* that learns relevant characteristics of participants based on observations of behavior and optimizes designs based on learned models. I prove theoretical properties of a method inspired by this framework, and demonstrate the feasibility and effectiveness of automated design procedures for automatically designing crowdsourcing tasks and synthesizing crowd workflows.

1.1 Crowdsourcing Complex Tasks

1.1.1 Human Computation and Crowdsourcing

Over the last decade, *human computation* [95, 52] has established itself as a powerful paradigm for incorporating human intelligence in problem-solving efforts in which machines cannot yet tackle the problem alone. Such systems take advantage of human abilities—e.g., in vision, natural language, and pattern recognition—to handle instances and aspects of problems that are difficult for computers. The ESP game [97], FoldIt [15], and reCAPTCHA [98] are a few examples of successful systems that draw on human contributors and machine computations to tackle problems in image labeling, protein folding, and text digitization.

Many human computation systems treat humans as processors in a distributed system, each performing a small part of a massive computation [95]. But unlike computers, humans require an incentive to contribute to a computational effort. This incentive may be in the form of monetary rewards, a sense of duty or purpose, or enjoyment of the task. For example, a human computation system may recruit a crowd of paid workers through an online labor market such as Amazon Mechanical Turk, in which workers receive small amounts of money for completing "microtasks." Another system may draw on a crowd of friends or followers on social media services such as Facebook and Twitter, who may be willing to contribute based on their relationship with the requester or other contributors. Yet another system may attract a crowd of users of a web service who are willing or required to contribute to a task, e.g., either because the task itself has been made enjoyable as in games with a purpose like the ESP game [97], or because completing the task is required for accessing content of value, as in reCAPTCHA [98].

Human attention is limited, and the incentive for individuals to contribute to a task is also limited. Practically, attracting a large crowd to perform an arbitrary task often implies that individuals in the crowd may only be involved briefly, and that any given individual in the crowd may provide noisy inputs. This is in contrast to the way work is performed in traditional firms and also in social computing systems such as Wikipedia, where employees and dedicated volunteers are available over time and can be relied upon to work on larger problems, keep track of context, identify issues, and solve problems that arise. To handle short periods of work and noisy inputs, human computation systems aim to break down large problems into smaller tasks, and provide means of quality control for synthesizing noisy inputs from large numbers of individuals.

1.1.2 Human Computation Algorithms

While simple tasks may be easy to parallelize across individuals, *complex tasks* require more sophisticated coordination and optimization. Over the last several years, there has been a rise of *human computation algorithms*, or *workflows*, that decompose a task into more manageable, self-contained subtasks. Human computation algorithms aim to allow individuals to contribute to small subtasks independently, without having to reason about other subtasks or the task at large.

By drawing on core computational principles, a number of *design patterns* have emerged that serve as the basic building blocks of human computation algorithms. For example, Little et al. [60] introduced an iterative design pattern in which each member of the crowd improves upon the previous solution. Zhang et al. [105], Bernstein et al. [5], and Kittur et al. [46] demonstrated how divide-and-conquer can be applied to the crowd in which crowd workers decompose problems, solve subproblems, and recompose subproblems into a solution. Methods for quality control can be incorporated at various points within an algorithm to promote high quality results.

Given a particular problem, crowdsourcing a solution may involve constructing a human computation algorithm that utilizes multiple design patterns. In Chapter

2, I will present an approach for formulating effective workflows by reasoning about characteristics of the crowd. I will demonstrate how this approach leads to design patterns, algorithms, and frameworks that combine existing design patterns to enable the crowd to reach expert level performance on complex tasks such as audio transcription and nutrition analysis.

1.1.3 Human Computation with Global Constraints

An important class of underexplored human computation tasks are those in which the solution must satisfy a set of global requirements. For example, in leveraging the crowd to write an essay, a requester may want to specify requirements on the desired tone, tense, length, structure of arguments, and style of exposition that must hold consistently throughout a piece of writing. Some requirements, e.g., presenting a balanced perspective on a situation, touch upon different components of the essay and depend on the essay as a whole.

As another example, consider the problem of crowdsourcing itinerary planning. Planning events such as vacations, outings, and dates often involve an *itinerary* which contains an ordered list of activities that are meant to be executed in sequence. People going on a trip have preferences and constraints over the types of activities of interest (e.g., “I am interested in history museums”), how long to spend on different activities (e.g., “I want to spend at least 2 hours hiking”), the composition of activities (e.g., “I want to focus on art galleries and museums for the day”), the budget, and the total time available, which define a set of global requirements that an itinerary should satisfy.

For these and other tasks that involve global constraints, good solutions rely on the composition of different contributions as a whole, with interdependence among solution components. As such, it is not clear how to break down these tasks into smaller tasks for individuals in the crowd to complete independently.

In Chapter 3, I will introduce *crowdware* as an approach for tackling human computation tasks with global constraints. Crowdware draws inspiration from *groupware* [25], which suggest principles and ideas on communication and collaboration within a shared context that help a group accomplish a joint task. Crowd workers differ from groups in that individuals may only be involved briefly, may be less willing to spend time grasping the solution context or taking meta-level actions, and may not fully consider the overall objective nor the aims of other crowd workers when making decisions. To address this challenge, crowdware provides mechanisms in which the system (indirectly) coordinates the problem-solving effort by focusing the crowd's attention on what needs work. I will present a system for crowdsourcing itinerary planning called Mobi, to illustrate this concept.

1.1.4 Harnessing Crowd Abilities: Control and Synthesis

Human computation algorithms tend to define an explicit sequence of steps in which individuals in the crowd are recruited to complete subroutines within this predefined process. However, there are also opportunities for the crowd to contribute to a problem-solving effort by guiding the control flow of an algorithm or even generating plans that define the problem-solving process. Taking this broader perspective, I explore in Chapter 4 different ways in which the crowd can contribute to a human computation process, leading to new applications and more efficient forms of problem solving.

An example of having the crowd guide the problem-solving process is *task routing*. Task routing aims to harness the ability of people to both contribute to a solution and to route tasks to others who they believe can effectively solve and route. Task routing provides an interesting paradigm for problem solving in which individuals become engaged with tasks based on their peers' assessments of their expertise. On the task level, effective task routing aims to take advantage of participants' individual expertise as well as participants' knowledge about others' abilities to contribute. On the organizational level, task routing can provide a means for bringing tasks to individuals effectively, where participants' routing decisions take into account not only an individual's expertise on a particular task, but also their ability to contribute as a router.

In Chapter 5, I will introduce incentive mechanisms that reward individuals for solving and routing tasks. An interesting problem that arises is that incentives need to take into account limitations on individuals' knowledge about the knowledge of others. For example, in a social network setting where individuals may only know about the expertise of those that are close to them in social distance (e.g., their friends, and possibly friends of friends), the would-be optimal incentive mechanism designed under the assumption that everyone knows everyone else's expertise may not work as desired. I will show how we can design incentive mechanisms that are sensitive to such limitations, while still promoting effective routing decisions.

1.2 Automated Environment Design

Many tools on the Internet facilitate data-driven, iterative design processes. Web analytics software tracks complex individual and group behaviors over time, and provides summary information on trends and patterns in the data. Frameworks, style sheets, and content management systems make it easier to modify or extend existing designs. Tools for A/B testing allow designers to compare alternative designs based on defined objectives.

Despite having a rich set of tools, identifying effective designs to elicit desired behaviors remains a process that is largely manual, tedious, and ad hoc. One potential solution is to automate the environment design process to systematically explore a design space in a principled, data-driven manner. Such an approach may be able to discover effective designs quickly, while requiring less manual effort. An automated environment design system may take as input a set of available interventions, the objective of the designer, and a model of the interaction among environment, participants, and behaviors, and provide as output an intervention that promotes actions and outcomes meeting the objective whenever such interventions exist.

One challenge in building such a system is that models of behavior are imperfect and incomplete. We have limited knowledge of users' preferences and decision-making processes, and this private information is difficult to elicit directly. But in online settings where the designer can track individual and group behaviors, such information can be indirectly inferred from observing actions over repeated interactions. For example, one can infer from observing consumer purchasing decisions and worker performance on tasks the underlying preferences and abilities that guide their decisions and the actions observed. These observations can be used to refine existing models and drive better design decisions.

While observing user behavior can provide some information, observations of user behavior under any particular design will only provide partial information about users' underlying preferences or abilities. For example, a consumer's purchase decision does not completely reveal their underlying value for a good (which may be at or above the purchase price), and a worker's performance on a task does not reveal exactly how the worker will perform on a different task. But given the ability to experiment with alternate designs and receive rapid feedback on different designs, it may be possible to iterate and refine our understanding of participants over time.

In Chapter 6, I will present a general approach for automated environment design that draws on these observations. I will provide an *active, indirect elicitation* framework that automatically drives an objective-based iterative design process by interweaving indirect learning of model parameters with optimization to determine effective interventions based on the current model. In Chapter 7 and 8, I will show how to apply ideas from automated environment design to automatically design crowdsourcing tasks and synthesize crowd workflows.

1.3 Limitations

Approaching computational environment design problems by reasoning and learning about characteristics of participants leads to solutions that are tailored to the participants. As such, specific results generalize only as far as the assumed characteristics of participants hold true across domains. For example, in addressing problems in crowdsourcing complex tasks, I propose designs that take into account some assumed characteristics of crowd workers (e.g., they may only make small contributions) and demonstrate that the designs are effective for participants satisfying such characteristics. But in other settings, these particular assumptions about the crowd may be false, and other characteristics of participants such as their intrinsic motivation may require very different designs.

Related to this, I assume throughout the dissertation that characteristics of participants remain more or less constant, and do not reason explicitly about how these characteristics may change over time or how potential changes may affect design decisions.

The automated environment design framework is most applicable for learning and reasoning about the characteristics of participants, and using this to parametrize designs, rather than for discovering effective design patterns in the first place. For complex domains, proposing a design space that includes effective designs may require significant amounts of domain knowledge. Automated design tools can help to refine existing models through experimentation and provide new insights, but are not yet a replacement for human ingenuity.

1.4 Thesis and Contributions

My thesis statement is:

> By reasoning and learning about characteristics of participants and how these characteristics interact with the decision environment to influence behavior, we can design environments that elicit desired actions and outcomes.

My contributions span several areas. Contributions related to crowdsourcing complex tasks include:

- Design patterns, algorithms, and frameworks that effectively utilize combinations of existing design patterns to enable the crowd to reach expert level performance on complex tasks. Constructed workflows coordinate the contributions from the crowd, and are effective in applications to crowdsourced audio transcription and crowdsourced nutrition analysis based on food photographs.

- A crowdware design pattern that enables a crowd to effectively resolve global constraints. Crowdware provides a shared, collaborative workspace through which individuals in the crowd contribute opportunistically based on the current solution context, and in which the system indirectly coordinates the problem-solving effort by alerting crowd workers to what needs work. This design pattern is demonstrated through a system called Mobi, in which the crowd generates custom itineraries for day trips.

- Methods and design elements that leverage the crowd's ability to control an algorithmic procedure and generate plans that define the problem-solving process. An experiment on the 8-puzzle that involves sharing problem-solving strategy, and a system called CrowdPlan that produces simple plans in response to high-level search queries, demonstrate that these methods and design elements can enable effective problem solving and novel applications.

- *Routing scoring rules* for prediction tasks that properly incentivize participants to jointly contribute to a task and route the task to others for further contributions. Theoretical results characterize the family of routing scoring rules that promote tractable routing decisions based only on local information.

Contributions related to automated environment design include:

- A model of the computational environment design problem.
- A general framework for active, indirect elicitation for automated environment design.
- Efficient algorithms for active, indirect elicitation in sequential decision making settings in which the principal can modulate costs and rewards, along with theoretical results about convergence.

Contributions that relate to both crowdsourcing and automated environment design include:

- Crowdsourcing applications that demonstrate the effectiveness of applying the automated design approach to designing an image labeling task and to synthesizing sorting algorithms tailored to crowd abilities.
- An automated design framework and associated learning and optimization algorithms for synthesizing crowd workflows.

1.5 Thesis Overview

This dissertation consists of two major components. The first component demonstrates how we can reason about crowd abilities and limitations to discover effective designs for crowdsourcing complex tasks:

- Chapter 2 introduces design patterns from the literature that serve as the basic building blocks for designing human computation algorithms. The chapter shows how to combine these design patterns to construct human computation algorithms for audio transcription and nutrition analysis.

- Chapter 3 shows how to tackle human computation tasks that are difficult to decompose. The chapter introduces Mobi, a system for crowdsourced itinerary planning. Mobi illustrates a novel *crowdware* design for tackling complex tasks with global constraints by using a shared, collaborative workspace. Experiments and user studies show that Mobi enables the crowd to effectively resolve violated constraints, and generates itineraries that satisfy users' stated requirements.

- Chapter 4 provides an overview of the different ways in which crowds can contribute to a problem-solving process by guiding the control flow of an algorithm and generating plans that define the problem-solving process. As examples, the chapter shows how passing around context can enable a crowd to more effectively solve a version of the 8-puzzle, and introduces a system called CrowdPlan, that leverages a crowd to generate simple plans for accomplishing high-level tasks.

- Chapter 5 describes methods for task routing that aim to harness people's ability to both contribute to a solution and to route tasks to others who they believe can further contribute. Focusing on prediction tasks, the chapter introduces routing scoring rules that reward effective contributions via solving and routing. The chapter also identifies a family of *local routing rules*, which isolate simple routing decisions in equilibria that are invariant to non-local information while still promoting effective routing and information aggregation.

The second component focuses on constructing automated procedures that can automatically derive effective designs by reasoning and learning about participants:

- Chapter 6 introduces a general approach for automated environment design and describes an active, indirect elicitation framework for iteratively refining designs. As an illustrative example, the chapter introduces the problem of *policy teaching*, in which the goal is to elicit a desired policy from a single agent in sequential decision domains modeled as Markov Decision Processes. Theoretical results provide conditions under which an algorithm applying the active, indirect elicitation framework is guaranteed to discover an effective intervention after a small number of interactions.

- Chapter 7 explains how to automate the design of human computation tasks. Using image labeling as an example, the chapter shows how to learn models of crowd performance as a function of design parameters and derive new designs by optimizing over learned models. Experimental results demonstrate that optimized designs collect significantly more high quality labels than baseline designs at the same rate of pay.

- Chapter 8 explains how to automatically synthesize crowdsourcing workflows. The chapter introduces an active, indirect elicitation based approach that selects experiments to refine current models of the crowd's performance on tasks in order to synthesize algorithms that optimize desired objectives given resource constraints. The approach is demonstrated to be effective through human sorting tasks that leverage the crowd to determine the ordering among objects.

Chapter 9 concludes the dissertation with a summary of contributions and a discussion of future research directions.

1.6 For the Reader

Most of this dissertation is intended to be readable by a general audience with interest in the design of social and economic Internet systems. The most technical chapters are 5, 6, and 8, where some familiarity with game theory (Chapter 5), decision science (Chapters 6 and 8), and artificial intelligence (Chapters 6 and 8) is assumed. Technical details in these chapters may be skipped with little loss to understanding the principles and ideas being introduced, nor the overarching message of approaching computational environment design problems through reasoning and learning about participants.

1.7 Bibliographic Notes

Chapter 2 includes work on iterative dual pathway structure [56] with Beatrice Liem and Yiling Chen, and work on PlateMate [69] with Jon Noronha, Eric Hysen, and Krzysztof Gajos. Chapter 3 presents work on Mobi [106] with Edith Law, Rob Miller, Krzysztof Gajos, David Parkes, and Eric Horvitz. Chapter 4 discusses work on crowdsourcing general computation [105] with Eric Horvitz, Rob Miller, and David

Parkes, and work on CrowdPlan [53] with Edith Law. Work on task routing [104] in chapter 5 was conducted with Eric Horvitz, Yiling Chen, and David Parkes.

Chapter 6 discusses work on automated environment design [103, 107, 108] with David Parkes and Yiling Chen. Chapter 7 presents work on automated task design [38] with Eric Huang, David Parkes, Krzysztof Gajos, and Yiling Chen.

Chapter 2

Human Computation Algorithms

In this chapter we study human computation algorithms that coordinate the efforts of a crowd to tackle complex tasks. A human computation algorithm is a set of instructions designed to be executed by humans and machines. Human computation algorithms can include function calls that are assigned to a crowd of individuals recruited to help contribute to the task. Depending on the task, one may recruit a crowd of paid workers through an online labor market such as Amazon Mechanical Turk, a crowd of friends or followers on social media services such as Facebook and Twitter, or a crowd of users of a web service who are willing or required to contribute to tasks.

Unlike employees in traditional firms or dedicated volunteers on Wikipedia who tend to be available over time, keep track of context, and generally provide good solutions, the types of crowds we can readily recruit may consist of individuals who are only briefly involved in any particular task (e.g., up to 10 minutes), and include individuals who may or may not provide helpful inputs. Given these characteristics, an effective human computation algorithm that calls on such crowds must break down complex problems into smaller tasks, and synthesize noisy inputs from large numbers of individuals to provide quality solutions.

The chapter is organized as follows. Section 2.1 introduces design patterns from the literature that serve as the basic building blocks for a human computation algorithm. Section 2.2 considers the problem of crowdsourcing audio transcription, and illustrates how to effectively combine existing design patterns to derive new design patterns and algorithms. Section 2.3 considers the problem of crowdsourcing nutrition analysis from food photographs. We demonstrate how to combine our understanding of how an expert performs a task with our understanding of the crowd to derive an effective workflow. Section 2.4 discusses how the ideas presented in this chapter may in general be applied to the design of human computation algorithms and tasks.

2.1 Design Patterns

2.1.1 Design Pattern 1: Divide-and-Conquer

A complex task can be too large for an individual crowd worker and thus require contributions from multiple individuals. To enable effective coordination among human problem solvers, we can draw on algorithm design patterns such as *divide-and-conquer*, which decomposes a problem into subproblems and composes solutions of subproblems into a solution. Divide-and-conquer algorithms are intended for parallel processing and are thus ideal candidates for human computation. Figure 2.1 depicts the decompose, solve, and recompose structure of divide-and-conquer algorithms.

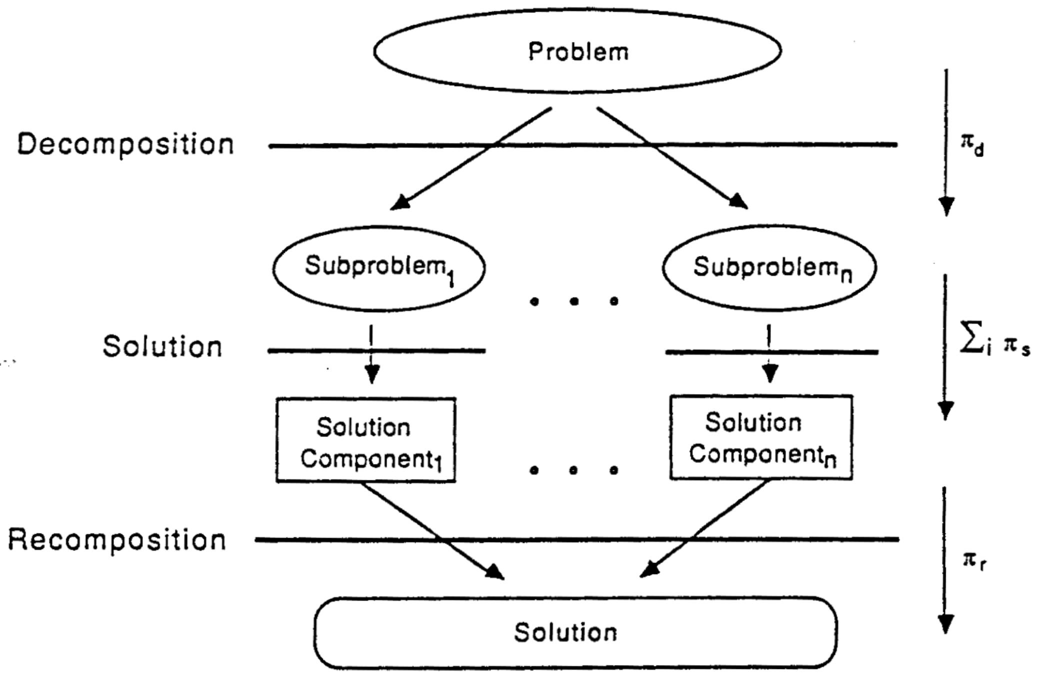


Figure 2.1: Diagram depicting the decompose, solve, and recompose structure of divide-and-conquer algorithms, with reasoning about costs and tradeoffs associated with different phases of the problem solving process. From Horvitz [34].

For distributed human computation, the decompose, solve, and compose steps may be performed by humans [105]. For example, Bernstein et al. [5] introduced Soylent, a system that uses a Find-Fix-Verify design pattern that is well-suited for open-ended tasks such as text editing. Soylent harnesses the crowd to identify patches of a document that need work (decompose), suggest potential fixes for each patch (solve), and filter out poor suggestions (recompose). By applying MapReduce, a programming framework based on divide-and-conquer, Kittur et al. [46] introduced a system called CrowdForge and constructed human computation algorithms for writing simple articles and making product comparisons.

2.1.2 Design Pattern 2: Redundancy-based Quality Control

Regardless of a task's size or design, there is no guarantee that any given individual that is assigned the task will provide a good answer. One way to approach this problem is to require *redundancy*, and find ways to synthesize noisy inputs. For example, consider asking a multiple choice question, such as what category a product should belong in. For this simple task, it is reasonable to assume that individuals putting in good faith effort are more likely to select the right answer than any particular wrong answer. By asking a sufficient number of people to perform the same task independently, we can take the most common answer as the solution, and expect with high probability that this is the actual, correct answer. Even if there are a few *spammers* who try to game the system by not completing the task in good faith, the most common answer is still likely to be correct if we collect a sufficient number of (non-spam) answers.

Over the years, researchers and practitioners have developed *quality control mechanisms* for human computation that aim to make efficient use of redundancy to guarantee high quality results. A quality control mechanism may choose a “best answer” from a set of answers directly, or recruit a crowd to vote on answers and decide based on collected votes. A mechanism need not necessarily choose the answer that is most common or most voted upon, and may for example weigh answers based on the presumed quality of workers as judged by their past work [40]. Quality control mechanisms can be employed at various points within a human computation algorithm, and deciding how much redundancy to use at any given point is often a tradeoff between the cost of effort required and the accuracy desired in the eventual result.

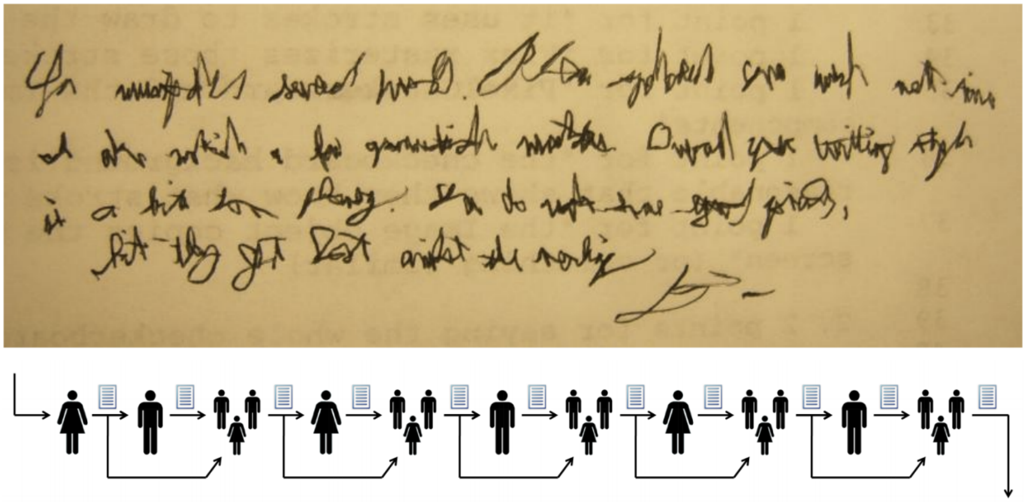
While it is important to be able to determine the correct answer from a set of collected answers, it is just as important to collect good answers in the first place. In addition to providing workers with proper instructions, examples, and training, requesters can attempt to elicit good faith efforts from crowd workers by rewarding correct answers. We do not normally know what the correct answer is, but we can use quality control mechanisms to both identify solutions that are likely to be correct and to reward contributors.

For example, in the ESP game [96], an image is displayed to two players whose goal is to reach an agreement on a label for the image. To encourage good faith effort, the ESP game uses an *output-agreement* mechanism [97] in which players are given the same input independently and are only rewarded for agreeing on an output. Since players are only likely to match on labels if they contribute in good faith, matching inputs can be used both to reward players and to identify relevant labels.

2.1.3 Design Pattern 3: Iterative Improvement

But despite any incentives we can provide or quality control mechanisms we can leverage, there are situations in which a task is inherently difficult and no individual working on the task independently is likely to correctly complete the task. As an example, Figure 2.2 shows a passage of poorly handwritten text, for which any individual transcribing this text may not be able to correctly decipher all the words.

As one approach for handling such problems, Little et al. [59] introduced an *iterative* design pattern, in which crowd workers are recruited to contribute sequentially to the same task. Each worker sees the task and the solution from the previous



"You (misspelled) (several) (words). Please spellcheck your work next time. I also notice a few grammatical mistakes. Overall your writing style is a bit too phoney. You do make some good (points), but they got lost amidst the (writing). (signature)"

Highlighted words should be: "flowery", "get", "verbiage", and "B-".

Figure 2.2: A passage of poorly handwritten text is transcribed using a human computation algorithm based on an *iterative* design pattern. Individuals in the crowd improve upon previous transcriptions. Voting tasks are used in between improvement tasks to decide whether a new solution is indeed an improvement over the previous. From Little [58].

worker, and is asked to improve upon that solution. To add quality control, iterative improvement steps can be interleaved with voting steps, in which crowd workers are recruited to judge whether the last revision is indeed an improvement over the previous solution. For problems such as transcribing poorly handwritten text, Little et al. [59] showed that iterative improvement can lead to higher quality solutions than the best solution from individuals in the crowd working on the task independently.

2.2 Case Study: Audio Transcription

Having introduced a number of design patterns for human computation algorithms, we consider how to apply these design patterns for solving actual problems. Solving any particular problem may benefit from utilizing multiple design patterns, and the goal is to discover effective and efficient ways of leveraging the crowd.

As a case study, consider the problem of audio transcription. There is a widespread need for transcription services that convert audio files into written text for a variety of purposes. Common examples include transcribing meeting minutes, court reports, notes for medical records, interviews, videos, and speeches. One benefit of having text is that it is easier to analyze and store than audio. Apart from this, there are many circumstances in which individuals rely on audio transcriptions in their daily lives. For example, a person who is deaf may wish to understand the audio content within a multimedia recording, and a person with limited ability to type, such as someone who suffers from carpal tunnel syndrome, may wish to create text documents.

Audio transcription is currently achieved mainly through two methods: professional human transcription and computer transcription. Professional transcription firms guarantee accuracies as high as 99% for fees as “low” as $1 per minute of transcribed text. Computer software presents a cheaper alternative but achieves significantly lower accuracies than professional human transcription. As humans are more adept than computers at deciphering speech and even non-professionals can contribute, crowdsourced audio transcription is being explored as a means for obtaining low cost, high-accuracy transcriptions. CastingWords is an example of such a service that recruits workers on Amazon Mechanical Turk (Turkers) to provide transcriptions, grade transcriptions, and improve transcriptions.[1] CastingWords charges between \$1 and \$2.50 per minute of audio transcribed depending on the required turnaround time, and pays workers based on the quality of the transcription and the task's difficulty.

One of the major challenges for crowdsourcing audio transcription is ensuring high transcription accuracy without knowing the correct answer. CastingWords has a fairly advanced quality control system that relies on Turkers to grade previous transcripts. To ensure that graders are putting in good faith effort, CastingWords uses a number of mechanisms for grade monitoring, such as grading the graders and using multiple graders to check a given clip.

While it is impressive that CastingWords is able to streamline their quality control system, all of the human effort spent on quality control does not directly help to improve the transcription accuracy. In this section, we design a human computation algorithm for audio transcription that eliminates the need for an explicit quality control process, focuses the crowd's effort solely on improving transcriptions, and achieves high transcription accuracy.

2.2.1 Designing an Algorithm

As people in the crowd may only be willing to spend a limited amount of time on tasks, we can apply the divide-and-conquer design pattern to break audio files into short, non-overlapping segments (decompose), obtain transcripts for these segments (solve), and rejoin these transcripts at a later time (recompose). Figure 2.3 shows

[1] `http://castingwords.com/`

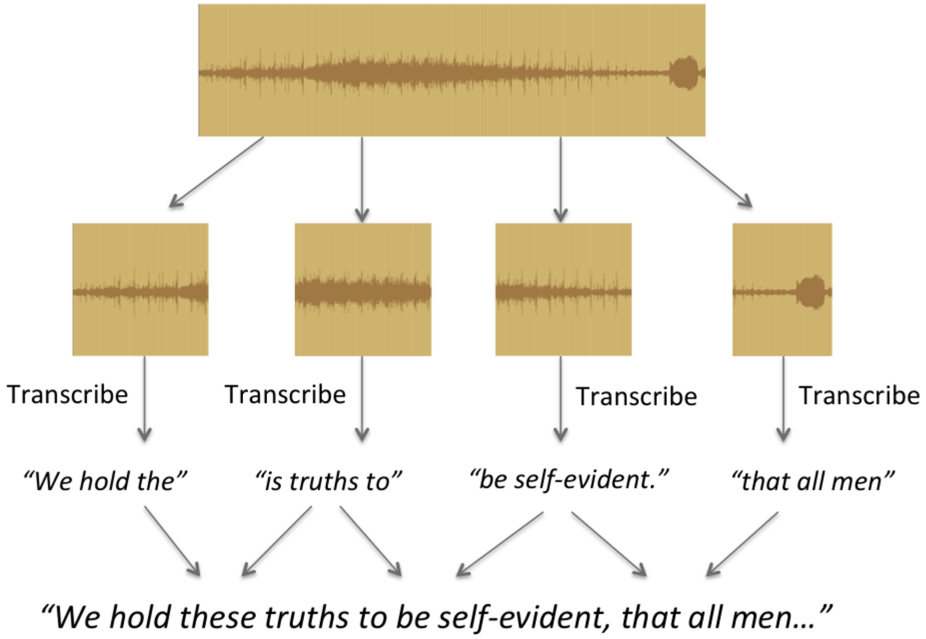


Figure 2.3: Divide-and-conquer is applied to crowdsourcing audio transcription. An audio recording is partitioned into shorter clips which are transcribed and then rejoined.

the high-level structure of an algorithm for crowdsourcing audio transcription based on this application of divide-and-conquer.

The decompose step can be done by the machine, but we will need the crowd to help with transcribing and rejoining transcripts. For a crowd of non-professionals, some audio clips may be difficult to transcribe correctly, e.g., due to background noise, speaker accent, or recording quality. Here we can apply an iterative design pattern: contributors are asked to transcribe to the best of their ability, and later contributors are asked to improve existing transcriptions by correcting any mistakes they encounter.

To rejoin transcripts of adjacent clips, we need to address the possibility that a word may have been split when the clip was initially divided. One solution is to combine two adjacent audio clips and present it along with the transcriptions of the adjacent audio clips for a contributor to join. To ensure that the person only edits where the two transcripts join, we can allow the person to only edit the transcript near the middle of a combined transcript. As rejoining transcripts can be a difficult task for anyone to complete correctly, we can again apply the iterative design pattern for this recompose step.

To implement the iterative design pattern for transcribing and rejoining transcripts, we still have to answer a couple of questions. First, how do we decide when to stop iterating? We can choose to iterate for a fixed number of steps, but it is not clear whether the solution after a fixed number of iterations would be a good one, or whether a good solution would have already been obtained after fewer iterations.

Second, how can we ensure that people are providing good inputs? While most people are likely to exert good faith effort, we would like to keep spammers out and to reward good solutions. We can include voting tasks after each iteration to check whether the last solution improves upon the previous, but for audio transcription this form of quality control is expensive. A person comparing two transcriptions may have to listen to the audio clip multiple times, and go back and forth to determine which transcription is better. As an alternative we can ask people to grade transcripts, but this is also expensive. While voting and grading are useful work, any human effort spent on quality control does not directly help to improve transcription accuracy, and may be better directed towards actually improving transcriptions.

To address these questions, let us take a step back from the iterative design pattern and think about what happens if we ask two people to transcribe the same audio clip independently. Intuitively, if both people exert good faith effort, they are likely to come up with “similar” transcripts, even if both transcripts may contain some mistakes. But if one or both people do not exert good faith effort, it is unlikely that the two transcripts will look similar. In this setting, an output-agreement mechanism can thus reward people for producing similar transcripts as a means to elicit good effort contributions. Given this observation, we would like to be able to design an algorithm that reaps both the benefit of eliciting good effort from output-agreement mechanisms and the benefit of improving transcription over time from iteration, while refraining from using an explicit quality control process.

To do this, we introduce an *iterative dual pathway structure*. For each clip, we assign contributors to one of two transcription pathways, alternating assignment by order of arrival. A contributor listens to the clip and sees recent transcripts submitted by previous contributors assigned to the same pathway (in our implementation, the last two transcripts). Each contributor’s submission is then compared to recent transcripts submitted by contributors on the other pathway (in our implementation, the last two transcripts), which he is never allowed to see. Because a contributor on one pathway is unable to see the transcripts produced by contributors assigned to the other pathway, the two paths should be independent. As contributors are expected to base their transcriptions on the contents of the audio file, we conjecture that the more similar the two pathways are, the more accurate they are.

In this structure, contributors’ submissions are scored based on their similarity to transcripts produced in the other pathway. If their contributions are vastly different, we can remove these results to avoid misleading future contributors or causing future

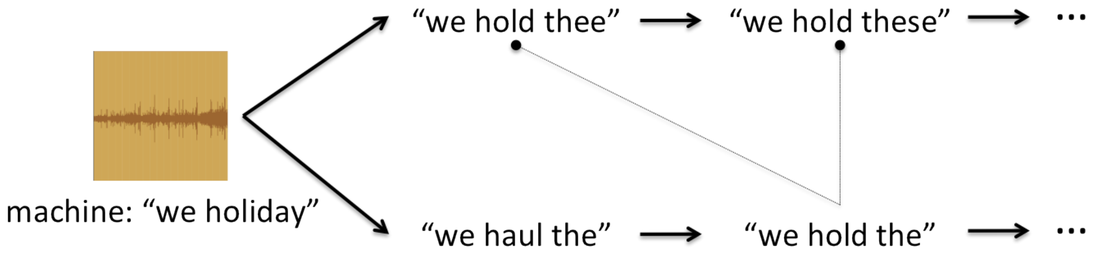


Figure 2.4: Contributors are alternately assigned to one of two pathways. They modify previous transcripts from their own pathway, and their transcripts are scored based on how well they match recent entries in the opposite pathway.

transcripts to be mis-scored. Comparing contributors' submissions to previous results necessitates having something to compare them to at the beginning; thus, at the start of the process, we can generate a computerized transcript of the audio file. This transcript can be treated as though it were produced by a previous contributor on the opposite pathway. It is used for comparison, but not for display and modification purposes.

Figure 2.4 shows an example of a clip being transcribed through the iterative dual pathway structure. We see that contributors modify previous transcripts from their own pathway, and their transcripts are scored based on how well they match recent entries in the opposite pathway.

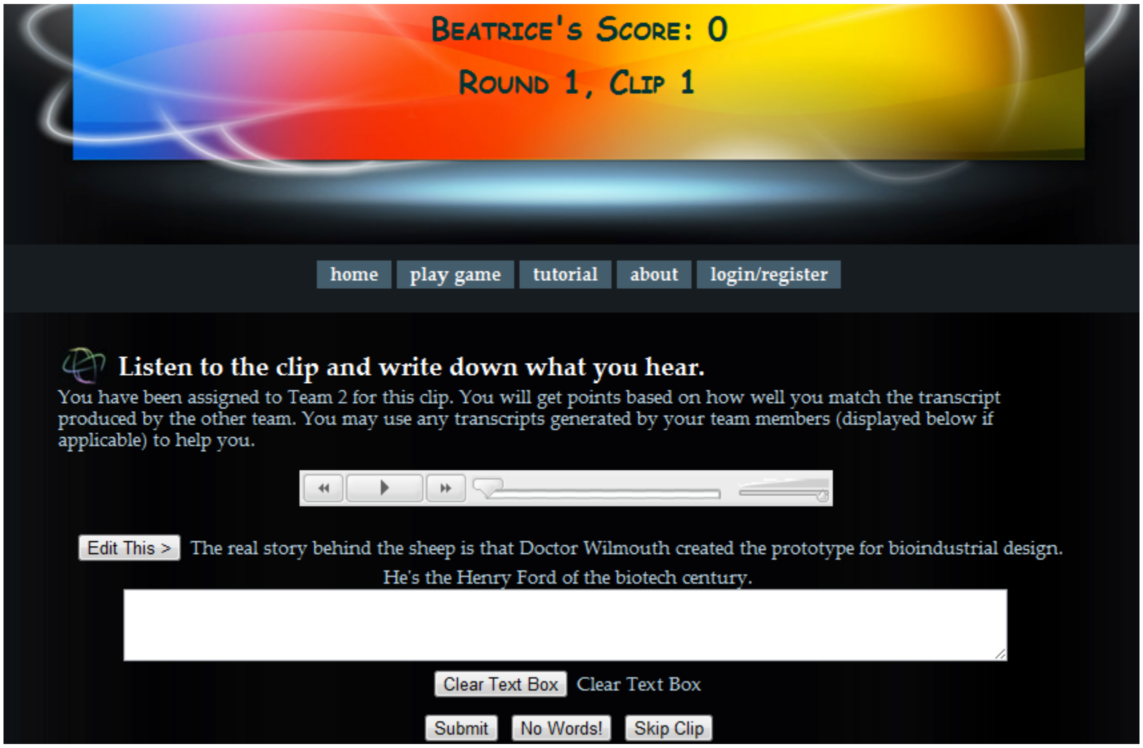
As contributors iteratively improve on previous results, transcripts should eventually converge to an accurate transcription of the content of the audio file. For example, we may decide to stop when four transcripts in a row (i.e., two from each pathway) match each other, at which point we deem that the clip has been correctly transcribed. Termination can thus be based on converging to the correct answer, and not rely on a fixed number of iterations determined a priori.

The iterative dual pathway structure has nice properties: it allows us to estimate the accuracy of a given transcript by comparing it to other transcripts (thus eliminating the need to check transcriptions in a separate process) and provides contributors with the proper incentives to enter accurate results. Because the two paths evolve independently and contributors can base their transcriptions only on the clips given to them, chances are that the more similar transcripts are, the more likely it is that they are close to being correct. By separating what contributors see from what they are being compared against for rewarding purposes, the dual pathway structure aligns incentives so that people are motivated to produce accurate transcripts. In doing so, *the iterative dual pathway structure effectively combines the output-agreement design pattern with the iterative design pattern to encourage contributors to provide accurate improvements.*

Putting it all together, we have a human computation algorithm for audio transcription that uses divide-and-conquer to break the task down into smaller tasks. By leveraging the iterative dual pathway structure, both the solving tasks (obtaining transcripts for short clips) and recomposing tasks (rejoining transcripts) simply ask people to improve on existing transcripts, and do not require explicit quality control.

2.2.2 Experiments

To test the effectiveness of our transcription algorithm, we recruited 147 Harvard University undergraduates to play an online game that implements our algorithm. Subjects were recruited through undergraduate housing mailing lists, and were offered a chance to win a $25 Amazon.com gift card. Subjects were told that the gift card



Listen to the clip and write down what you hear.

You have been assigned to Team 2 for this clip. You will get points based on how well you match the transcript produced by the other team. You may use any transcripts generated by your team members (displayed below if applicable) to help you.

The real story behind the sheep is that Doctor Wilmouth created the prototype for bioindustrial design. He's the Henry Ford of the biotech century.

Figure 2.5: A screenshot of the user interface

is given to one person chosen at random, where each person's chance of winning is directly proportional to the total number of points accumulated through gameplay.

For our experiments, we obtained clips from `http://www.americanrhetoric.com/`, most of which came from movies and speeches. Clips ranged in length, clarity, content matter, and the degree to which they used uncommon words, proper nouns, and slang. Clips were passed through Adobe Soundbooth CS4 (transcribed on High Quality, using American English) to produce the computer transcripts that seeded the iterative dual pathway structure.

Figure 2.5 provides a screenshot of the user interface for the iterative dual pathway version of the game. In this version, players transcribed each clip and were awarded points according to how closely their transcripts matched the transcripts of players on the opposite path. The similarity between transcripts was measured using the *Levenshtein distance* [54], which measures the number of insertions, deletions, and substitutions on a character basis between two strings. The game ran for a week from 3/7/2011 to 3/14/2011. We used 20 audio files, for a total of 44 shorter ten-second clips and 25 longer 20-second clips that spanned these shorter clips. Players produced 549 transcripts over the course of gameplay.

In addition to the iterative dual pathway version of the game, we also implemented a *parallel* version for comparison. The parallel implementation did not allow players to see what others entered. Players were asked to transcribe the clip from scratch, and players' entries were scored randomly. This implementation consisted of 10 audio files divided into 20 ten-second clips. Longer clips were not created for this experiment, so the accuracy reported here only reflects that of the ten-second segments. The parallel implementation of the game also ran for a week, from 2/26/2011 to 3/6/2011. Players produced 308 transcripts.[2]

To compare our results to industry figures concerning transcription accuracy, we used *word accuracy* (WAcc), which is measured as a percentage and calculated on a word basis as follows:

$$\text{WAcc} = 100 * (1 - \frac{\text{Insertions} + \text{Deletions} + \text{Substitutions}}{\#\text{ of Words In Accurate Transcript}}) \tag{2.1}$$

For other evaluations, we used a variation of word accuracy which we call *character accuracy* (CAcc). This metric computes accuracy using the Levenshtein distance (LD)

[2] While players were allowed to participate in more than one version of the game, players were never allowed to transcribe the same clip in both two versions.

on a character basis as follows:

$$\text{LD} = \text{Insertions} + \text{Deletions} + \text{Substitutions} \tag{2.2}$$

$$\text{CAcc} = 100 * (1 - \frac{\text{LD}}{\#\text{ of Chars In Accurate Transcript}}) \tag{2.3}$$

We found that in all cases tested, word accuracy and character accuracy were comparable.

Overall, the word accuracy for the parallel process was 93.6%, compared to 96.6% for the iterative dual pathway process. The latter accuracy is comparable to the accuracy advertised by professional transcription. The accuracy of the clips that converged for the iterative process was 97.4%, compared to an average of 95.5% for those that had not. Given more time and additional iterations, it is likely that the 96.6% accuracy we found would have been higher; in many instances, errors came not in the middle of transcripts, but across breaking points between clips where fewer iterations were completed.

Figure 2.6 shows the average across all clips of the minimum, average, and maximum character accuracies of transcripts in the two pathways after k iterations (i.e., after k contributors in each path have transcribed a clip). We find that the minimum and average accuracies increased over time, and the difference in the maximum and minimum accuracies between the two clips decreased. This indicates that as the number of iterations increased, clips became more similar and more accurate.

Table 2.1 shows the number, percentage, cumulative percentage, and accuracy of clips that converged after exactly k iteration. Also shown is the accuracy level across all clips that converged after exactly k-th iteration. We see that many clips converged early on and that accuracies do not appear to depend on when a clip converged.

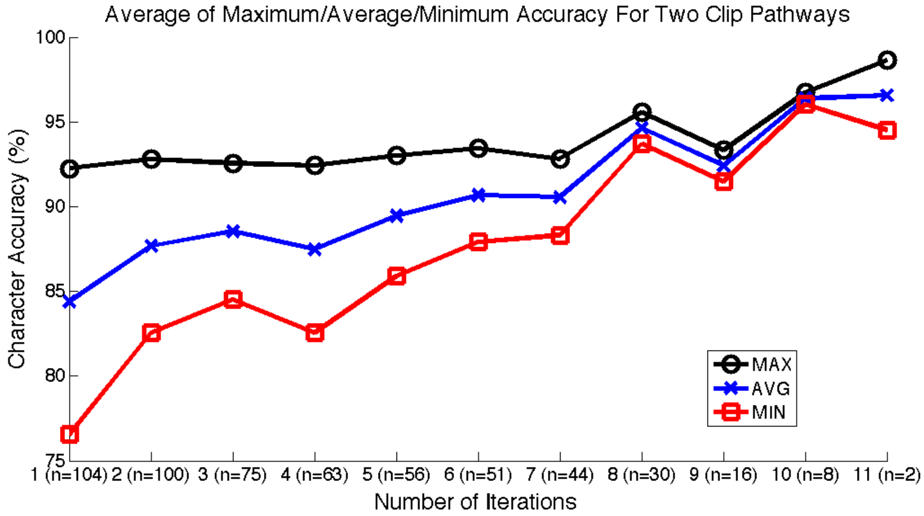


Figure 2.6: The average maximum/average/minimum accuracies of transcripts in the two pathways after k iterations. Transcripts are removed from this graph after they converge to avoid an upward bias.

We also compared the enjoyability and efficiency of the iterative dual pathway structure against that of the parallel structure. By surveying participants, we found that players enjoyed the iterative dual pathway structure more than the parallel one; they liked correcting clips more than transcribing them anew, and they played the former game longer than the latter. Additionally, as would be expected, players spent less time processing clips in the iterative process than in the parallel one, with mean transcription times of 33.1 seconds and 39.5 seconds respectively. The mean transcription time for the iterative dual pathway tasks even includes transcription of the 20-second clips. These results suggest that the iterative dual pathway structure is more enjoyable and more efficient than the parallel one.

Analyzing specific transcriptions and the ways in which they evolved provide evidence that the iterative process was fairly successful in allowing players to correct

Iter.	**# Conv.**	**% Conv.**	**Cumul. %**	**WAcc (%)**
2	11	21.2	21.2	96.0
2.5	5	12.5	31.4	100.0
3	3	8.6	37.3	100.0
4	1	3.3	40.8	100.0
4.5	1	3.4	42.9	90.0
5.5	1	3.7	45.8	95.7
6.5	1	4.2	50.0	100.0
7.5	3	15.8	61.9	95.6
8	1	8.3	71.1	95.7

Table 2.1: Number and percentage of clips that reached k iterations, cumulative percentage of clips converging before or reaching k iterations, and word accuracy of clips converging in the k-th iteration. Iterations where no clips converged are not displayed. Half-number iterations refer to an uneven number of transcripts on each path (i.e., 2.5 iterations means that one of the two paths had two iterations while the other had three).

others' misspellings or decipher additional portions of the clip. Here is one such example showing corrections made in the early iterations:

Iteration 1 red, red, red! what should i do?

Iteration 2 red, red, red! Dear God, where should I go, what should i do?

Iteration 3 Fred, Fred, Fred! Dear God, where shall I go, what should i do?

Iteration 4 Rhett, Rhett, Rhett! Dear God, where shall I go, what shall I do?

(*"Dear God" should be "If you go"*)

Players were also adept at rejoining clips:

Beginning Transcript You have modernized your economy, harnessed your rivers, diversified your industry, liberalized your trade, electrified your fa arms, accelerated your rate of growth. (*Break in the middle of "fa arms"*)

Iteration 1 You have modernized your economy, harnessed your rivers, diversified your industry, liberalized your trade, electrified your farms Accelerated your rate of growth.

Iteration 2 You have modernized your economy, harnessed your rivers, diversified your industry, liberalized your trade, electrified your farms, accelerated your rate of growth.

(*Correct*)

2.2.3 Summary

We introduce a new human computation algorithm for audio transcription. In the process of deriving the algorithm, we demonstrated how different design patterns (divide-and-conquer, iterative improvement, output agreement) can be effectively utilized together to solve a problem. In doing so, we also discovered a novel iterative dual pathway structure that combines the benefits of output agreement and iterative improvement, eliminating the need for explicit quality control in iterative workflows.

2.3 Case Study: Nutrition Analysis

In the audio transcription algorithm, human effort is elicited only for providing transcripts. In general, human computation algorithms can harness different types of human effort and coordinate the inputs and outputs of different tasks. For some problems, the algorithmic challenge is identifying what types of effort to elicit, and effectively coordinating heterogeneous contributions to derive a solution. In this section, we demonstrate how to combine our understanding of how an expert performs a task with our understanding of the crowd to derive an effective solution for crowdsourced nutrition analysis.

The majority of Americans perceive healthy eating as complicated [20]. For people who commit to changing their eating habits, accurate logs of what they eat may help in monitoring progress toward set goals [61]. Currently, food logging is typically done by hand using paper diaries, spreadsheets, or a growing number of specialized applications. This process is time-consuming and error-prone [74, 27]; a review of nine studies found error rates from -76% (underestimates) to $+24\%$ (overestimates) [82]. A number of online interfaces exist to simplify the process, but they still require tedious logging that discourages recording. Studies have also found that self-reports using these interfaces are no more accurate than pen and paper [3, 102].

Martin et al. [65] suggested an alternative approach called the Remote Food Photography Method (RFPM). Rather than typing names of foods and estimating portions, users are asked to photograph their plates at the beginning of the meal and at the end to accurately capture how much food was actually eaten. Trained dietitians identify the pictured foods remotely and estimate portions. The results of laboratory studies showed that dietitians using RFPM underestimated calories by only 5-7% compared to directly weighing the foods [65].

RFPM thus combines the accuracy of direct observation by experts with the convenience of free-living conditions. Users of the method found it satisfying and easy to use [65]. The problem is cost and scarcity. RFPM relies on experts to analyze each photograph, limiting the system's accessibility and potential scale. The method might be feasible in specific healthcare settings, but trained dietitians are too costly and scarce for general use.

This suggests an opportunity for crowdsourced nutrition analysis. Prior research indicates that the most difficult part of nutrition analysis is estimating portion size [65], and that trained amateurs have low bias but high variance [64]. The "wisdom of crowds" is ideally suited to these situations, since the average of amateur estimates often beats a single expert [91].

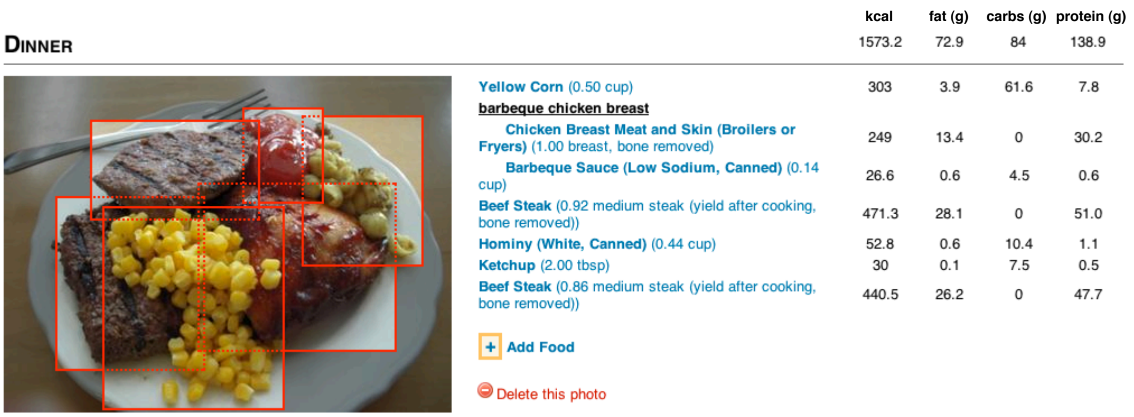
A recent iPhone application demonstrates, however, that naive approaches to crowdsourcing for nutrition analysis are not sufficient. In April, 2011, the fitness website Daily Burn released Meal Snap, which allows users to photograph foods and receive calorie estimates by so-called "pure magic."[3] Meal Snap creates a single Mechanical Turk task for each image. Workers provide a free text description of food, and the application appears to match this description with a database of average consumption to estimate a range of possible calories. This approach is appealing, but critics have accused it of failing to provide accurate data.[4]

2.3.1 PlateMate

To make accurate food logging easier and more affordable, we introduce PlateMate, a system for crowdsourcing nutrition analysis from photographs of meals using Amazon Mechanical Turk. PlateMate allows users to upload food photographs and receive nutrition estimates within a few hours. The estimates consist of a list of foods

[3] `http://mealsnap.com/`, accessed July 5, 2011

[4] `http://www.mobilecrunch.com/2011/04/05/too-lazy-to-count-calories-now-you-can-just-take-a-picture-of-your-meal/`



DINNER	kcal	fat (g)	carbs (g)	protein (g)
	1573.2	72.9	84	138.9
Yellow Corn (0.50 cup)	303	3.9	61.6	7.8
barbeque chicken breast				
Chicken Breast Meat and Skin (Broilers or Fryers) (1.00 breast, bone removed)	249	13.4	0	30.2
Barbeque Sauce (Low Sodium, Canned) (0.14 cup)	26.6	0.6	4.5	0.6
Beef Steak (0.92 medium steak (yield after cooking, bone removed))	471.3	28.1	0	51.0
Hominy (White, Canned) (0.44 cup)	52.8	0.6	10.4	1.1
Ketchup (2.00 tbsp)	30	0.1	7.5	0.5
Beef Steak (0.86 medium steak (yield after cooking, bone removed))	440.5	26.2	0	47.7

+ Add Food

Delete this photo

Figure 2.7: The PlateMate user interface. Users upload photographs of their meals, which are processed through Mechanical Turk to produce a list of foods, serving sizes, and nutrition information.

in the photograph, with associated measurements of serving size, calories, fat, carbohydrates, and protein for each food item. The information is displayed to the user via the interface shown in Figure 2.7.

Crowdsourcing nutrition analysis presents several challenges in task and workflow design. First, Turkers are inexperienced, and may produce unreliable estimates. Second, most Mechanical Turk tasks are simple and Turkers may be unaccustomed to performing complex tasks like nutrition analysis. Finally, any individual Turker may be biased in their estimates or have trouble recognizing certain foods contained in a photograph.

To best design a workflow for crowdsourcing nutrition analysis, we started by observing a dietitian as she determined nutritional data from several photographs. Her process consisted of three distinct steps: identifying foods in each image, estimating their portions, and then calculating the corresponding nutrition data. The final step can be fully computerized, but PlateMate implements the first two with crowdsourc-

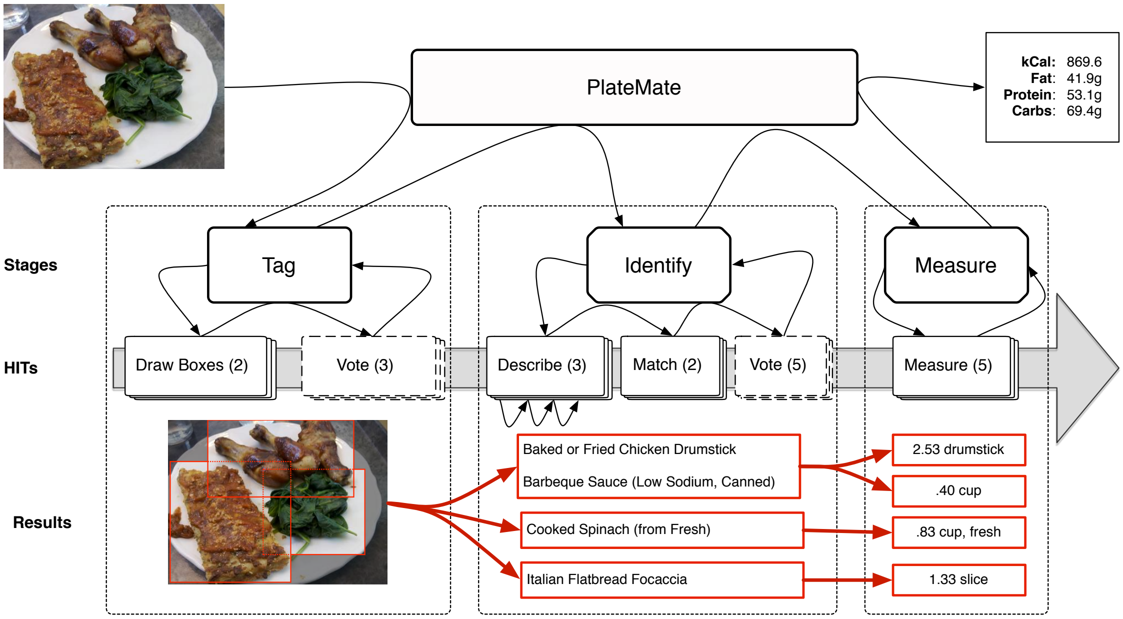


Figure 2.8: The PlateMate system. Work travels between stages and Human Intelligence Tasks (HITs) along the black arrows, starting from the input on the left and concluding with the output on the right. The system takes submitted photos and creates Tag tasks to annotate these photos with boxes. Each box becomes the input to a series of Identify tasks which end with a list of foods from a commercial food database. Each individual food is then passed to a Measure task, which produces a unit and amount. Dashed boxes represent optional stages, which may be skipped during routing.

ing. To parallelize work, we add an input decomposition stage at the start in which the crowd partitions a plate into distinct foods.

The result is a workflow with three major stages, shown in Figure 2.8. *Tag* takes photos and labels them with boxes drawn around distinct foods on a plate. *Identify* matches each box to one or more foods in a commercial nutrition database. *Measure* returns portion estimates for each identified food.

Step 1: Tag

The goal of the Tag stage is to find every food item in a photograph. One picture may depict several plates, and each plate may contain several distinct foods. Tag discovers these foods and distinguishes them by drawing a rectangle around each. The result is a group of boxes overlaid on the picture. Each box corresponds to a single food item, like a sandwich.

This step brings about a number of benefits. First, results can surface more naturally in the user interface. This makes estimates easier to understand and correct. Second, parallel work can also be combined more carefully, since we know which identifications describe each pictured food. Finally, the Tag step encourages completeness, and prevents workers from ignoring or forgetting to match certain foods.

Drawing Boxes: Tag's first Human Intelligence Task (HIT) asks workers to draw a box around each food in the picture. Workers need cultural background knowledge to understand how foods on a plate fit together. Pure computer vision can detect edges and boundaries, but may not recognize that an open-faced hamburger with the top half of the bun off to the side is in fact one item. The HIT relies on Turkers' general intuition about food items, and provides sandwiches, salads, and pasta with vegetables as examples of individual food items.

Similarity Comparison and Voting: Two Turkers are asked to tag each photo. Once both assignments are completed, they are algorithmically compared based on the number, size, and position of boxes. If the two groups are sufficiently similar, one is picked at random as the final answer. If the box groups differ significantly, three additional Turkers are shown each set overlaid on the photo and asked to select the better option, using similar guidelines. The box group receiving more votes is returned as the final result.

Step 2: Identify

The Identify step matches a tagged box to one or more food entries in a commercial nutrition database. While each box output from Tag should only contain one food item, some composite items do not exist in the database. For example, if "ham and cheese sandwich" is missing, Identify should choose "wheat bread," "sliced ham," and "American cheese."

There are two main challenges in this stage. Identifications must be correct, and when several correct identifications exist, the most compact one should be used in order to simplify measurement and eventual presentation of data to end users.

In pilot study, Identify was performed in a single HIT. Workers used an autocomplete text input to list each food in the box. Their answers were frequently incorrect or incomplete. Workers appeared to type a one-word description of the picture, like "chicken," and then select the first option regardless of the closeness of fit. Like the "Lazy Turkers" mentioned by Bernstein et al. [5], they performed the minimal work necessary and nothing more.

These problems may also have occurred because the interface asked Turkers to perform two conceptually different tasks sequentially but only produce one final output. Turkers first had to identify each food and then locate the corresponding entry in the database. To correct for this, we developed a workflow that contained two simpler HITs. The first asks workers to describe the food in their own words. The second asks (other) workers to match this description to items in the database.

Describing Items: In this HIT, Turkers see a box on a photo. One question asks "What is this food?" Here we request one-line descriptions like "pepperoni pizza" or "salad with chicken." The other question asks "What is it made of?" Here we provide a free-form text field where workers can list component parts. For simple foods like broccoli these fields will be identical, but for composite foods the fields should collect different answers that are each useful.

Following successful prior experiments by Little et al. [60] that have workers iteratively improve on descriptions of images, we also made this step iterative. One worker starts from blank fields. His answer becomes input to another HIT, where the next Turker is asked to improve on it by correcting mistakes and adding detail. This process is well-suited to the "Eager Beavers" mentioned by Bernstein et al. [5], who provide minute details and list many possibilities. It also handles "Lazy Turkers" well, since terse descriptions are progressively expanded.

Matching Foods: After three iterations, the output of the Describe task is fed into a Match HIT. Here, workers see the photo (with the box) and the final descriptions. They are asked to select the best entry or set of entries in the database to match the boxed portion of the photo, with the descriptions serving as suggestions for what to search. Workers first attempt to locate the description of the box as a whole in the database. If they do not find a good match, they search for each part. For example, workers should first search for “salad with chicken and tomatoes.” If this fails, they should look for “chicken breast,” “romaine lettuce,” and “cherry tomatoes.”

The search interface is modified from a standard autocomplete. Search results display below the input box, but the keyboard cannot be used for quick selection. Instead, Turkers must click on items to add them. The interface also makes it clearer that multiple items can be selected through several searches. These changes negate the instinct of “Lazy Turkers” from the pilot study to select the first item they see.

This decomposition makes each step manageable for Turkers moving through the HITs rapidly. The results of the Describe step are not necessary for the end goal of calculating nutrition information, but the generated descriptions reduce the mental work required for the Match step. We can then ask Turkers working on Match HITs to find the simplest representation in the database, using the Describe results as a guide.

Agreement Detection and Voting: Two workers are asked to complete each Match HIT. If each returns a list pointing to the exact same item or items in the food database, then that list is used. Otherwise, five workers complete a Vote HIT to decide between them.

Step 3: Measure

The Measure step produces an estimated portion size for each food matched in Identify. With these measurements, the nutrition data for a photo can be calculated by multiplying the per-unit nutrition breakdown from the food database.

Measure uses only one HIT, which shows Turkers a photo with a box highlighted along with the name of one food in that box. They are asked to first select a measurement unit and then provide a numeric estimate in terms of that unit. The units provided by the food database are specific to each food. “Pepperoni pizza” includes options like “slice, large” or “whole pie, medium,” while “white rice, cooked” uses cups or ounces.

Measurement is considered the most difficult step of this process for amateurs [65], so the Measure stage uses a number of techniques to produce accurate results. Presenting multiple measurement options is helpful, since many of these only require counting rather than estimating a weight or volume. For example, it is much easier to count florets than to estimate grams of broccoli.

Not every food can be measured by counting. To help in cases where weight or volume estimates are necessary, HITs include a portion guide which provides common approximations for different measurements. For example, three ounces of meat looks like a deck of cards and a quarter cup is roughly the size of a golf ball. These approximations are more error-prone than simple counting, but they allow workers to estimate portions without any training.

The interface also alerts Turkers to avoid making common errors. Pilot testing revealed that measurements in weight were much less accurate than those using volume or counting, so a warning is presented when Turkers choose grams, ounces, or pounds. Testing also indicated that some workers misunderstood the serving types. For example, for “chicken nuggets,” one worker selected “serving, 6 nuggets” and then entered 6 as the value. This indicated 6 servings of 6 nuggets each for 36 total.

To reduce these errors, the interface generates a calorie estimate on the fly and asks workers to eyeball their answer. They are given common calorie ranges for different meals and shown warnings if the count becomes unusually low or high. These warnings cannot prevent all errors, but they encourage Turkers to double-check their answers.

Aggregating Measurements: Five Turkers are presented with Measure HITs. The results from these HITs can be compared in the common units of calories. This means estimates can be aggregated without any additional human computation like voting. Drawing on the principle that averaging many high variance but low bias estimates can lead to accurate results [91], we remove outliers and then return the mean of the remaining estimates.

Turker Qualifications

When recruiting workers from an online labor market like Mechanical Turk, our algorithm decides on which workers to recruit by requiring workers to meet specified qualifications. After several iterations during pilot testing, we decided to accept only Turkers located in the United States who had previously completed at least 200 HITs and had a 98% HIT acceptance rate. As we planned to test the system on food photographs from the United States, we decided to require American Turkers due to the cultural context required for most elements of the process.

2.3.2 Evaluation

We evaluate the accuracy of estimates from the PlateMate system by comparing its crowdsourced estimates to current alternatives. Nutritional data returned by PlateMate was compared with ground truth, expert dietitian estimates, and estimates by a recent commercial application. A reader interested in additional user studies and analysis on the PlateMate system can refer to Noronha et al. [69].

The evaluation has two goals. The first was to determine the accuracy of PlateMate with ground truth data obtained from manufacturers or preparers. The second was to compare PlateMate's performance with two alternative approaches to remote food photography: analysis by experts and results from Meal Snap. Because Meal Snap only returns calorie information, and to make the task manageable for our expert participants, we limited our comparison to estimated calories even though PlateMate generates reports that also include fat, protein, and carbohydrates.

Method

We conducted the experiment with a sample of 18 photographs showing 36 distinct foods. Some photographs depicted individual foods or packages, while other photographs showed complex plates containing many items (see Figure 2.9). Each pictured food had nutritional data available through the manufacturer or preparer, and foods were weighed when necessary to ensure accuracy. These foods were selected to span a variety of meals and sources, including restaurants, cafeterias, and grocery items. We also included a mix of simple foods and composite items like salads and sandwiches.

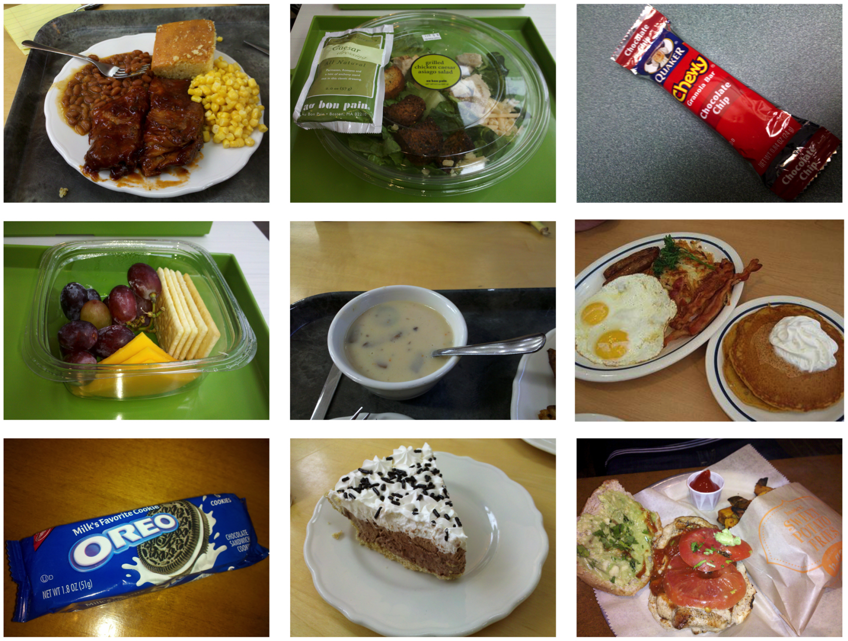


Figure 2.9: Examples of photos from the study of PlateMate's accuracy.

We recruited three professional dietitians to provide expert estimates: one was a private nutrition counselor, and the other two were hospital-based. They received compensation for their time and provided estimates from their own offices. They were encouraged to use any resources (e.g., books and calorie databases) that they would typically use for a similar task.

Our third set of estimates came from Meal Snap, a recent commercial iPhone application. Meal Snap creates a single Mechanical Turk task for each image. Workers provide a text description of food, and the application appears to match this description with a database of average consumption to estimate a range of possible calories. Meal Snap returns a range of calories rather than a definitive answer, so we used the mean of its high and low values.

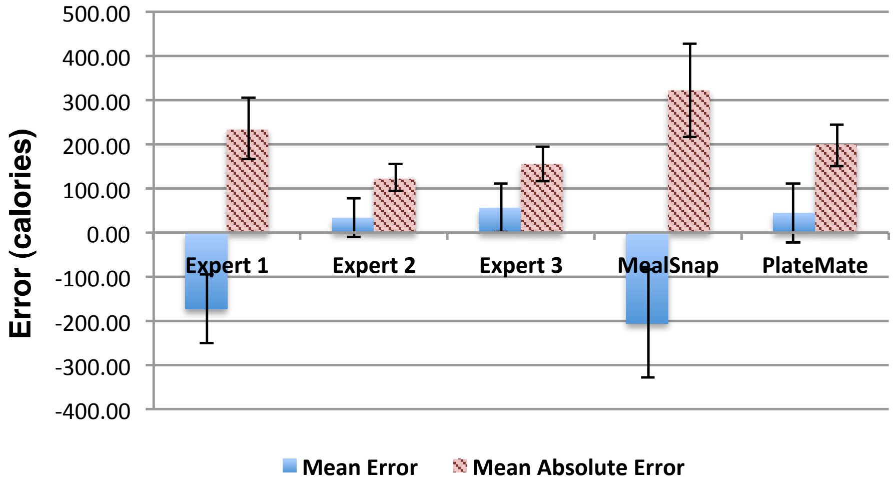


Figure 2.10: Mean errors (i.e., overall bias) and mean absolute errors (average magnitude of an error) for estimates made by the human experts, the Meal Snap application, and PlateMate compared to data provided by manufacturer or preparer. Error bars represent standard errors.

Results

In terms of mean absolute error on calorie estimates, PlateMate was not significantly different from the human experts or the Meal Snap application. Figure 2.10 illustrates the results in detail. As expected, trained dietitians were the most accurate on average. Their mean absolute error rates were 39.4%, 20.8%, and 26.1%, for an average of 172.0 calories or 28.7% per photograph. The best expert was off by just 124.5 calories, on average. PlateMate was close behind with a mean absolute error rate of 198 calories, or 33.2%. Meal Snap was farther behind, with an average error rate of 322.8 calories or 53.9%.

Absolute error rates reflect the average magnitude of the error, but not the biases in each method. To understand how estimates from each source would add up over time, we also measured mean error without taking absolute values. The best expert overestimated by just 32.75 calories on average, for a mean error rate of +5.5%. The other two experts had error rates of +9.2% and −27.5%.

In comparison, PlateMate had a mean error rate of +44.1 calories, or +7.4%, which was much closer than Meal Snap's −34.4%. Expert and PlateMate results are significantly correlated with the ground truth data ($r^2 = .8626$, .9062, and .9378 for the experts, and $r^2 = .8622$ for PlateMate, all with $p < .0001$), while there was no significant correlation between Meal Snap results and the ground truth data ($r^2 = .2352$, $p = .3475$).

PlateMate's error rate compares favorably to amateur self-reports, where error rates can be greater than 400 calories/day and range from −76% to +24% [82, 10]. It also lacks the systematic bias towards underestimation in self-reports, especially among vulnerable users. These results indicate that PlateMate's answers, while imperfect, can be a useful nutritional guide.

Error Analysis

Most errors in the study corresponded to single failures in specific parts of the pipeline. In the Tag stage, boxes were sometimes drawn improperly, leading to missing or duplicate identifications. In one photo of a brownie and banana on a small plate, only one box was drawn covering the entire banana and most of the brownie. As a result, the workers at the Identify stage omitted the brownie.

Most errors occurred in the Identify stage. Turkers had trouble distinguishing similar types of a food, which sometimes had large nutrition differences. A plate of vegetarian baked beans was identified as regular baked beans, tripling the calorie count. Branded foods also caused problems: a relatively low-calorie chicken sandwich was identified as a sandwich from the restaurant Chili's, which had over twice as many calories.

During measurement, very small quantities were often overestimated, especially when a small amount of a food was spread over a large area. Other errors occurred when one food appeared in several boxes. This led to a hamburger bun being counted as two buns when each half of the bun was seen in its own box.

These errors suggest areas for further improvement. In particular, introducing personalization and geolocation capabilities can help address many of the common errors we encountered. For example, we can adapt the task interface to emphasize the foods most common in a user's diet and thus most likely to appear in their photos, potentially leading to more accurate results. Geolocation capabilities available in many mobile devices could be used to further improve accuracy, especially for restaurant meals. Photos could be annotated with the cuisine of the restaurant in which they were taken, providing Turkers with helpful context while keeping the user's location private. Integrating with existing local "check-in" applications like Foursquare would make it even simpler to associate meals with their places of origin.

2.3.3 The Management Framework

Because PlateMate relies primarily on dividing human work into a number of heterogeneous and interacting tasks, and because the issues of worker skill and motivation were central to our design process, we found it conceptually helpful to use human organizational hierarchies as a metaphor for designing our system. Specifically, we observe that in the real world, expert-level work can sometimes be reproduced by less skilled workers—each working on a specific part of the process—supervised by managers who are not necessarily skilled craftsmen themselves, but who know how to assign tasks, route work among workers, and verify the quality of the work.

To implement division of labor for complex crowdsourcing tasks like nutrition analysis, we created a new framework organized around objects called *managers.* Managers communicate with their supervisors and their *employees* via asynchronous message passing: managers assign tasks by placing them in inboxes of lower level managers and communicate with their superiors by placing results of completed tasks in their own outboxes. This hierarchical message-passing approach allows programmers to implement workflows by decomposing problems into progressively smaller steps.

As illustrated earlier in Figure 2.8 (page 42), the root of this tree is a *chief manager*, which gathers new inputs and produces completed outputs. In PlateMate, the chief has three employees: Tag, Identify, and Measure. Each of these are in turn managers and have their own employees, corresponding to the individual HITs described above.

This hierarchical structure creates a flexible workflow consisting of modules connected by higher-level managers. Managers can route work intelligently among their employees, and may dynamically alter the sequence of steps in the process depending on a situation. For example, PlateMate's Tag manager compares the outputs from its DrawBoxes employee. If they are sufficiently different, they are sent to the VoteBoxes manager to decide between them. Otherwise, one answer is chosen randomly and sent up the hierarchy as Tag's completed output. All managers work in parallel, each processing its own stream of work.

When multiple tasks are submitted, processing is done just-in-time: for example, as soon as one photograph is tagged, the Identify manager begins the process of finding out what foods are present in each of the boxes without waiting for the remaining photographs to be tagged.

At the lowest level of the hierarchy are managers whose employees are the crowd workers. Managers at this level create jobs (such as asking for the food in one tagged box on a photo to be identified) and receive responses. Programmers create HIT templates and validation functions which are used by the framework to create HITs and approve work. Managers simply assign work to the crowd and receive validated outputs that can be passed up the tree.

The management approach differs conceptually from prior work, which has focused on creating "crowd programming languages" that combine human and machine computation. For example, TurKit [60] lets requesters program crowds in JavaScript, Qurk [63] integrates crowds into SQL, and CrowdForge [46] parallelizes work with MapReduce scripts. In each case, the toolkit attempts to make working with crowds more like working with computers. This approach emphasizes computation as the natural glue for combining individual worker contributions, and the resulting artifact is a computer program with some of the primitive operations implemented as functional calls to human workers. Of course, the Management Framework *is* a computational framework, and it naturally supports a number of design patterns for programming the crowd. For example, the Tag step is an analog of the decompose step in divide-

and-conquer, and the Describe step (part of Identify, see Figure 2.8) relies on iterative refinement [59] to improve the level of detail of the descriptions.

Management is implemented as an extension of Django, a web application framework for Python. It builds on several useful features from Django, including an HTML template language for defining HIT instructions, examples, and interfaces. It also uses Django's object-relational mapper, which automatically stores Python objects in a MySQL database. This means that the precise state of the system is always stored, including managers' inboxes and outboxes, active HITs and completed assignments, and intermediate inputs and outputs. This simplifies later analysis, since requesters can go back and query responses from each stage in the workflow. It also protects completed work from program errors or service outages; after crashes, execution simply resumes from the last good state.

2.3.4 Summary

We present PlateMate, a human computation system for nutrition analysis based on food photographs. In the process of deriving the PlateMate algorithm, we observed the steps that an expert took in approaching the problem, transformed these steps into stages of crowd problem solving, and decomposed each stage into smaller subtasks so that the crowd can contribute effectively. We also introduced the management framework inspired by the structure of human organizations, which provides effective support for managing crowdsourcing of complex heterogeneous tasks.

2.4 Discussion

Having formulated effective human computation algorithms for audio transcription and nutrition analysis based on food photographs, we present some general lessons and ideas on how to approach the design of human computation algorithms for tackling complex tasks.

Identify what needs to be done

Solving many complex tasks require identifying what types of effort to elicit, and effectively coordinating among heterogeneous contributions to derive a solution. To construct algorithms for the crowd, we may start by drawing inspiration from explicit workflows used by individuals or within organizations, or capturing the implicit steps an expert takes in the process of problem solving, as we did for PlateMate. This provides a sense of the different components that may go into a human computation algorithm, as well as the dependencies among these components.

Apply design patterns to overcome crowd limitations

Any tasks we identify need to be transformed into tasks that the crowd can effectively contribute to. This requires, for example, taking into account that individuals in the crowd may only work for short periods of time, and can generate noisy solutions. This transformation can be guided by identifying design patterns for overcoming any crowd limitations that make it undesirable to assign a task directly. For example, we used the divide-and-conquer design pattern for audio transcription to break down the task into transcription tasks with shorter clips whose transcriptions are later rejoined, and combined the iterative design pattern with the output-agreement design pattern to encourage participants to provide accurate improvements.

Design tasks based on the way the crowd works

To enable workers to make effective contributions without necessarily understanding the overarching goal or how different steps in a workflow contribute to that goal, each task should be self-contained, and designed such that the process of arriving at a good solution is conceptually simple. In initial pilot testing for PlateMate, we saw how using an autocomplete input box when asking workers to identify foods made it more straightforward for workers to select generic descriptions that would have led to inaccurate results. Reasoning about the process through which crowd workers perform a task, and being sensitive to how different workers may approach a task (e.g., some may be overeager while others may be lazy [5]), can lead to more effective designs.

Prototype early and run pilot experiments

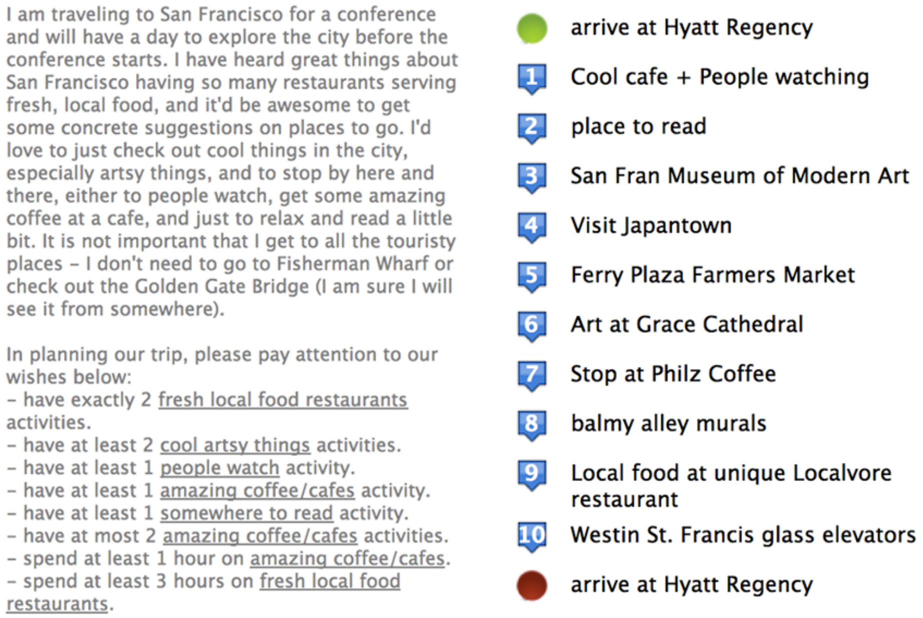
Last but not least, while we are often able to come up with the structure of an algorithm by reasoning about the crowd, we do not always know how the crowd will react to a particular task's design. In the nutrition analysis example, we used pilot studies to better understand how the crowd worked on tasks in order to tune task interfaces, instructions, and feedback to workers. Given the ease with which a requester can prototype and test their algorithms on a platform like Mechanical Turk, experimenting with alternative designs and using observations of workers to derive new designs can be a valuable tool for promoting helpful contributions.

Chapter 3

Human Computation with Global Constraints

Within studies of human computation, an important class of underexplored tasks are those in which the solution must satisfy a set of global requirements. For example, in leveraging the crowd to write an essay, a requester may want to specify requirements on the desired tone, tense, length, structure of arguments, and style of exposition that must hold consistently throughout a piece of writing. Some requirements, e.g., presenting a balanced perspective on a situation, touch upon different components of the essay and depend on the essay as a whole. Similar considerations arise in creative tasks such as graphic design and more mundane tasks such as meeting scheduling.

As good solutions rely on the composition as a whole and are marked by interdependence among solution components, tasks involving global constraints are difficult to decompose into subtasks that can be independently assigned to individuals in the crowd. This raises a significant challenge for human computation algorithms, as such



I am traveling to San Francisco for a conference and will have a day to explore the city before the conference starts. I have heard great things about San Francisco having so many restaurants serving fresh, local food, and it'd be awesome to get some concrete suggestions on places to go. I'd love to just check out cool things in the city, especially artsy things, and to stop by here and there, either to people watch, get some amazing coffee at a cafe, and just to relax and read a little bit. It is not important that I get to all the touristy places – I don't need to go to Fisherman Wharf or check out the Golden Gate Bridge (I am sure I will see it from somewhere).

In planning our trip, please pay attention to our wishes below:
– have exactly 2 fresh local food restaurants activities.
– have at least 2 cool artsy things activities.
– have at least 1 people watch activity.
– have at least 1 amazing coffee/cafes activity.
– have at least 1 somewhere to read activity.
– have at most 2 amazing coffee/cafes activities.
– spend at least 1 hour on amazing coffee/cafes.
– spend at least 3 hours on fresh local food restaurants.

● arrive at Hyatt Regency
1 Cool cafe + People watching
2 place to read
3 San Fran Museum of Modern Art
4 Visit Japantown
5 Ferry Plaza Farmers Market
6 Art at Grace Cathedral
7 Stop at Philz Coffee
8 balmy alley murals
9 Local food at unique Localvore restaurant
10 Westin St. Francis glass elevators
● arrive at Hyatt Regency

Figure 3.1: Planning mission (left) and itinerary (right)

tasks are not amenable to the divide and conquer approach introduced in chapter 2 that is used in most crowdsourcing systems.

As a focal example, consider the problem of crowdsourcing itinerary planning. Planning events such as vacations, outings, and dates often involve an *itinerary* (Figure 3.1), which contains an ordered list of activities that are meant to be executed in sequence over the course of an event. People going on a trip have preferences and constraints over the types of activities of interest (e.g., "I want a coffee break right after lunch"), how long to spend on different activities (e.g., "I want to spend at least 2 hours in parks"), the composition of activities (e.g., "I want to focus on art galleries and museums for the day"), the budget, and the total time available, which define a set of global requirements that an itinerary should satisfy.

Decisions on any particular activity in the itinerary may naturally influence other decisions. As simple examples, spending time on one activity leaves less time for another, and moving to one location introduces distances to other locations.

To handle tasks with global requirements, we introduce in this chapter a *crowdware* design that provides a single workspace in which a crowd of individuals contribute opportunistically based on their knowledge and expertise and the current solution context, and in which the system (indirectly) coordinates the crowd problem-solving effort by focusing the crowd's attention on what needs work. Crowdware takes inspiration from *groupware* [25], which suggest principles and ideas on communication and collaboration within a shared context that help a group to accomplish a joint task. We consider how to apply such principles and ideas to crowd workers, who differ from groups in that individuals may only be briefly involved, may be less willing to spend time grasping the solution context or take meta-level actions, and may not consider the desires of other crowd workers when making decisions.

We focus on itinerary planning as a case study of coordinating a crowd to tackle tasks with global constraints. We introduce a collaborative itinerary planning system called Mobi. Mobi takes a planning mission containing a set of qualitative and quantitative constraints as articulated by the user as input and produces an itinerary that satisfies the mission as output. The crowd participates via a single interface—displaying the current itinerary and a stream of ideas generated thus far—that allows individuals to contribute opportunistically given the current context and to see their contributions incorporated into the solution in real-time. Mobi focuses the crowd's attention on aspects of the evolving plan that needs work by prominently displaying a list of automatically generated *todo items*, which point out violated constraints, provide suggestions on how to address them, and promote activities directed at the refinement of the itinerary.

Mobi allows users to specify their desires and needs in *natural language*, thereby enabling complex constraints and preferences to be expressed and used in the planning process. We present two studies, which show that Mobi's design promotes a collaborative planning environment in which the crowd can effectively produce custom itineraries that satisfy the global constraints stated in user missions.

In the first study, we test the effect of displaying todo items on the rate at which quantitative constraints are resolved by the crowd, and measure the contribution patterns of crowd workers. We find that the display of todo items promotes satisfaction of constraints at a significantly faster rate than when todo items are not displayed, and that the crowd's editing patterns show evidence of both collaboration and opportunistic planning. In the second study, we seek to understand whether the end users believe that crowd-generated itineraries satisfy their stated requirements. Users report that the itineraries contain many activities of interest, mostly or fully satisfy their mission requirements, and are useful for their actual trips.

The chapter is organized as follows. Section 3.1 presents related work. Section 3.2 introduces the Mobi system. Section 3.3 and Section 3.4 describe our two studies. Section 3.5 revisits the elements of Mobi's design and discusses how these elements may in general inform the design of systems that facilitate a crowd to tackle problems involving global constraints. Section 3.6 summarizes our results and presents directions for future work.

3.1 Related Work

Planning can be viewed as an *iterative task* in which workers make successive edits to improve the solution. There has been some attention on iterative tasks in human computation [60], and an interesting recent example is work by Kittur [45] that recruits workers to collaborate in Etherpad to translate a poem. Workers were able to see their edits reflected in real time and could communicate via chat to explain their edits. One difference in Mobi is that Mobi uses its sense of the progress made so far (e.g., how full the itinerary is, which constraints are violated, etc.) to prompt users on what needs work so as to guide the problem-solving process.

Wikipedia can be viewed as an example of a system in which (mostly expert and highly dedicated) contributors write and edit articles to resolve a set of global constraints as defined by Wikipedia's standards. Much like the way todo items are used in Mobi to drive progress, template messages and cleanup tags are used in Wikipedia to alert editors of changes that need to be made to improve an article.[1] Such messages are typically managed by human contributors, whereas in Mobi todo items are managed in an automated manner whenever possible.

Several models have been proposed to describe how people generate plans to achieve goals. The *successive refinement* model advocates a top-down approach, where a high-level goal is decomposed into subgoals iteratively, down to a sequence of elementary actions [81]. In contrast, the planning of many everyday activities (e.g., errands) is often *opportunistic*. In other words, planning decisions happen whenever opportunities arise [30, 44], so that a decision or observation in one part of the plan

[1] See `http://en.wikipedia.org/wiki/Wikipedia:Template_messages/Cleanup`

may suggest new ideas or illuminate problems in a different part of the plan, causing the planner to refocus his attention. Opportunistic planning may involve both top-down and bottom-up processing. For example, in an errand planning experiment, Hayes-Roth and Hayes-Roth [30] found that subjects would start making detailed plans (e.g., sequencing individual errands), and then switch to planning on a more abstract level (e.g., by discovering clusters of errands), and back and forth as they refined the plan. Mobi is designed with the opportunistic planning model in mind, where individuals in the crowd are allowed to contribute freely as they see fit based on their observations of what needs work given the current solution context.

Real-life planning is a difficult problem for computers. Despite advances in automated planning [67], a major challenge is making sense of people's goals, preferences and other "soft" considerations [13]. Currently, the automated planner in Mobi supports workers by automatically checking constraints and computing trip times and routes. In the future, automation may play a more active role in the planning process by learning about different requirements, suggesting activities and their composition in the itinerary, or even detecting and adding important constraints that may have been missed by the requester.

There are several existing commercial systems that allow groups to plan trips for themselves or to ask friends and other members for suggestions. Examples include Gogobot, Triporama, Kukunu, and FriendTripper. Mobi differs from these systems in that it produces not only suggestions for activities, but an itinerary satisfying a set of global requirements. By using todo items, Mobi can also focus the crowd on making contributions where they are most needed.

3.2 Mobi: A System for Crowd Itinerary Planning

Mobi takes a planning mission consisting of preferences and constraints as input, and generates an itinerary by having a crowd plan asynchronously using a shared interface. Workers invited to contribute can view the current plan and all ideas proposed thus far, and make contributions as they see fit. Edits can be made at any time and without restrictions. The itinerary is automatically saved after each change. We now describe Mobi's interfaces for specifying the planning mission and assembling the itinerary, and discuss how these two interfaces support the process of generating itineraries and resolving constraints.

3.2.1 Specifying the Planning Mission

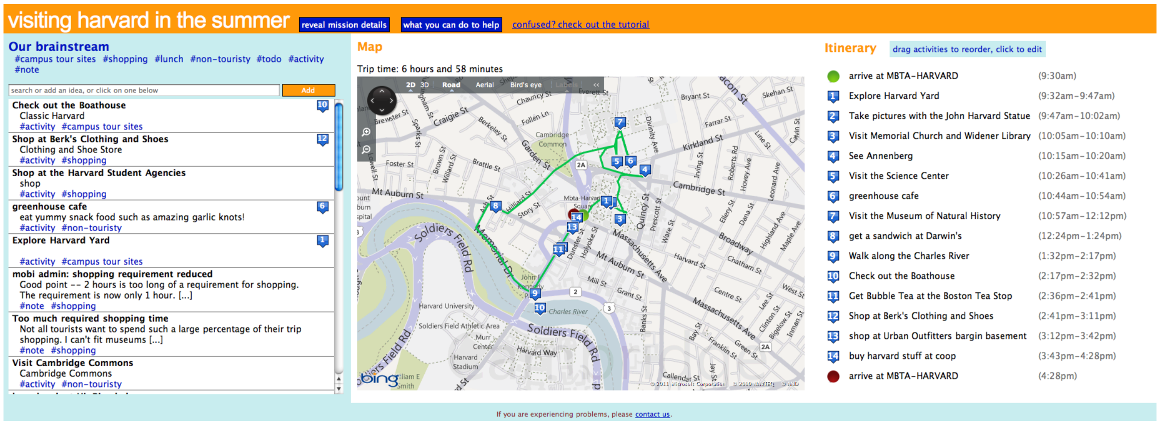
Our target users, also referred to as *requesters*, are people who are interested in planning a trip. To start planning, the requester enters a planning mission using a simple web interface, by specifying the title and description of the trip, start/end locations and times, and whether he or she will use public transit or drive between locations in addition to walking.

Requesters can express two kinds of constraints: *qualitative* and *quantitative*. Figure 3.1 (page 60) shows an example of a planning mission that includes both types of constraints. Qualitative constraints are specified in natural language (e.g., in a paragraph). They can describe, for example, the nature of the trip, what the user hopes to accomplish, and who they are traveling with. Quantitative constraints are specified by creating *categories* using arbitrary natural language phrases (e.g., "cool art," "by the ocean"), and assigning preferences and limitations over categories. One can specify constraints on the number of activities in each category (e.g., "I want to visit up to two *museums*"), as well as the amount of time to spend on activities in each category (e.g., "I want to spend at least two hours on *cool art*"). Such constraints can also be used to express the preferred combination of activities in the plan (e.g., "I want to spend half of my time on activities *by the ocean*, and the other half on activities *in the city*"). In our prototype, the domain-specific language for quantitative constraints allows for constraints encoded in the form of "*I want* {*at most, at least, exactly*} *[number]* {*activities, hours*} *of* $\{cat_1, cat_2, \ldots, cat_n\}$," where cat_i refers to the i-th requester-defined category.

Both qualitative and quantitative constraints contain natural language, and can express "soft" considerations that the computer cannot tackle alone. In addition to these constraints, the system maintains a pair of time constraints, which state that the cumulative duration of the activities in the itinerary should not be greater than, or significantly less than, the duration of the trip specified by the user.

3.2.2 Assembling the Itinerary

Once a requester specifies a planning mission, workers can use Mobi's planning interface to view the mission by clicking on the "reveal mission details" button in the *information panel* (Figure 3.2, on top). The planning interface consists of two key components: the *brainstream* and the *itinerary viewer*.


visiting harvard in the summer
reveal mission details
what you can do to help
confused? check out the tutorial
Map
Trip time: 6 hours and 58 minutes
2D 3D Road Aerial Bird's eye
bing
If you are experiencing problems, please contact us.

Our brainstream
#campus tour sites #shopping #lunch #non-touristy #todo #activity #note

search or add an idea, or click on one below — Add

Check out the Boathouse 10
Classic Harvard
#activity #campus tour sites

Shop at Berk's Clothing and Shoes 12
Clothing and Shoe Store
#activity #shopping

Shop at the Harvard Student Agencies
shop
#activity #shopping

greenhouse cafe 6
eat yummy snack food such as amazing garlic knots!
#activity #non-touristy

Explore Harvard Yard 1
#activity #campus tour sites

mobi admin: shopping requirement reduced
Good point -- 2 hours is too long of a requirement for shopping. The requirement is now only 1 hour. [...]
#note #shopping

Too much required shopping time
Not all tourists want to spend such a large percentage of their trip shopping. I can't fit museums [...]
#note #shopping

Visit Cambridge Commons
Cambridge Commons
#activity #non-touristy

Itinerary drag activities to reorder, click to edit

●	arrive at MBTA-HARVARD	(9:30am)
1	Explore Harvard Yard	(9:32am-9:47am)
2	Take pictures with the John Harvard Statue	(9:47am-10:02am)
3	Visit Memorial Church and Widener Library	(10:05am-10:10am)
4	See Annenberg	(10:15am-10:20am)
5	Visit the Science Center	(10:26am-10:41am)
6	greenhouse cafe	(10:44am-10:54am)
7	Visit the Museum of Natural History	(10:57am-12:12pm)
8	get a sandwich at Darwin's	(12:24pm-1:24pm)
9	Walk along the Charles River	(1:32pm-2:17pm)
10	Check out the Boathouse	(2:17pm-2:32pm)
11	Get Bubble Tea at the Boston Tea Stop	(2:36pm-2:41pm)
12	Shop at Berk's Clothing and Shoes	(2:41pm-3:11pm)
13	shop at Urban Outfitters bargin basement	(3:12pm-3:42pm)
14	buy harvard stuff at coop	(3:43pm-4:28pm)
●	arrive at MBTA-HARVARD	(4:28pm)

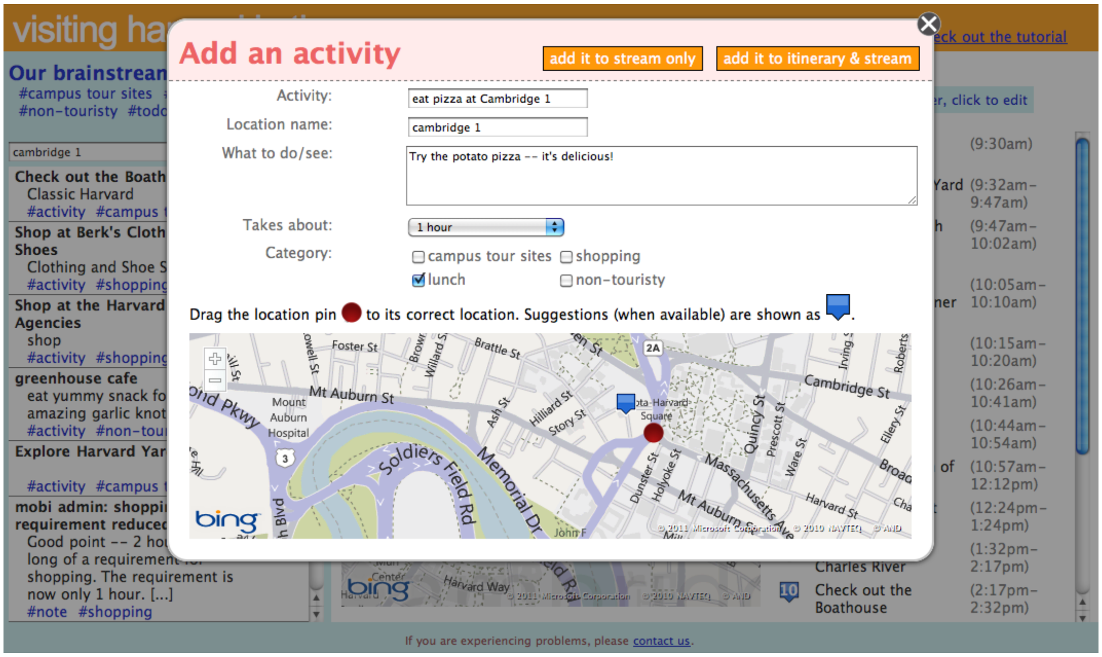
Figure 3.2: The Mobi planning interface consists of the information panel (top), the brainstream (left), and the itinerary viewer (right).

Brainstream

The *brainstream* (Figure 3.2, on left) is a collection of everyone's ideas. An idea can be an activity ("something to do or see") or a note ("a thought about the plan").

To view ideas in the brainstream, one can either scroll down the list, click on a hashtag to display ideas belonging to a particular category, or use the autocomplete search box. Clicking on an idea reveals a dialog box with additional details, an option to edit the idea, and in the case of an activity, an option to add it to or remove it from the current itinerary. A blue badge next to an activity indicates that it is already in the current itinerary.

To add a new idea (an activity or a note), one can type a title into the search box and click "add." If similar ideas already exist, a drop down list will appear, which helps to prevent duplicates and promote editing. For notes, workers can fill in a description. For activities, the *activity editor* (Figure 3.3) asks workers to provide the name of the location, what to do or see, the activity's duration, and the (requester-



Add an activity

add it to stream only | add it to itinerary & stream

Activity: eat pizza at Cambridge 1

Location name: cambridge 1

What to do/see: Try the potato pizza -- it's delicious!

Takes about: 1 hour

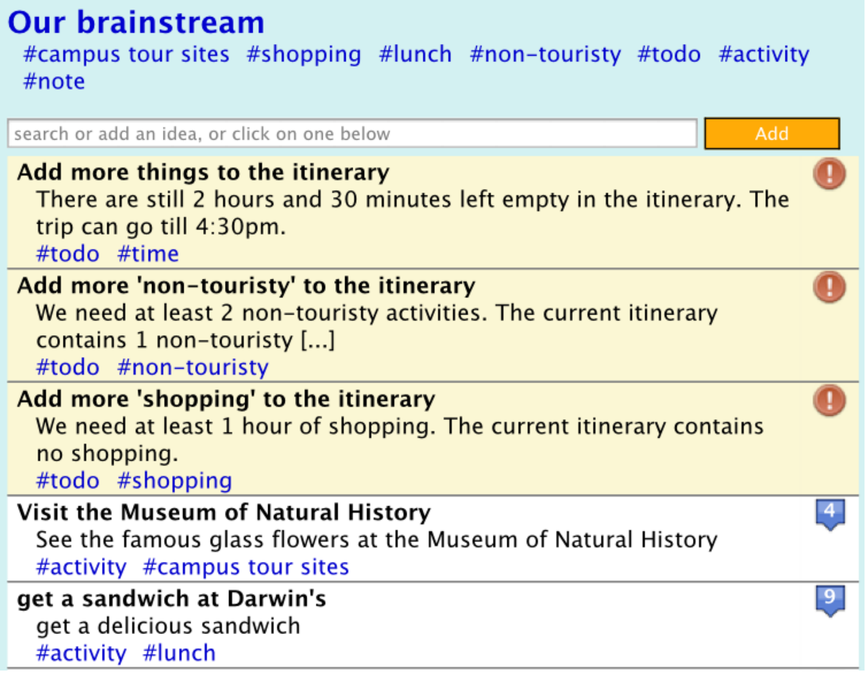
Category: □ campus tour sites □ shopping ☑ lunch □ non-touristy

Drag the location pin ● to its correct location. Suggestions (when available) are shown as ▾.

Figure 3.3: Adding a new activity to the brainstream

defined) categories that the activity belongs in. In the same editor, workers can view a map, which allows them to mark the location of the point of interest. Workers can decide to add the activity to both the itinerary and the brainstream, or only to the brainstream for the time being.

The brainstream allows workers to brainstorm together and build upon each other's ideas. It keeps around all suggested activities, and allows workers to quickly access them through the hashtags and the search box. By adding notes, workers can identify areas that need work or raise questions about the plan's feasibility, which other workers or the requester can then help to address or provide comments on. The brainstream's design draws inspirations from social technologies such as Twitter and Piazza, that aggregate information into a feed or stream that one can easily process.



Our brainstream

#campus tour sites #shopping #lunch #non-touristy #todo #activity #note

Add more things to the itinerary
There are still 2 hours and 30 minutes left empty in the itinerary. The trip can go till 4:30pm.
#todo #time

Add more 'non-touristy' to the itinerary
We need at least 2 non-touristy activities. The current itinerary contains 1 non-touristy [...]
#todo #non-touristy

Add more 'shopping' to the itinerary
We need at least 1 hour of shopping. The current itinerary contains no shopping.
#todo #shopping

Visit the Museum of Natural History
See the famous glass flowers at the Museum of Natural History
#activity #campus tour sites

get a sandwich at Darwin's
get a delicious sandwich
#activity #lunch

Figure 3.4: The brainstream displays system-generated todo items which alert workers to what needs work.

If the current itinerary does not satisfy a quantitative constraint or is over time or under time, the violated constraints are automatically turned into *todo items* that are displayed at the top of the brainstream with exclamation marks (Figure 3.4). Todo items alert workers to what needs work. They suggest specific actions, such as “Add a ‘lunch’ activity” or “The itinerary is over time. Try reordering itinerary items. You can also edit or remove items.” Todo items also provide natural language explanations of how the current itinerary violates particular constraints. For example, a todo item may explain that “You need a ‘lunch’ activity but there is currently none in the itinerary” or “The itinerary is over time because the trip must end by 9pm.”

We note that the system is able to check arbitrary quantitative constraints and generate todo items without understanding the meaning of the natural language categories. This is because workers associate activities with the categories they belong in when the activities are suggested. As we will show in the next section, todo items are an important design element that accelerates the speed at which quantitative constraints are resolved.

Itinerary Viewer

The *itinerary viewer* (Figure 3.2, on right) consists of an itinerary and a map. The itinerary displays the activities in order, with times during which they are scheduled to take place. Travel times between locations are automatically computed and accounted for. The map displays the activities' locations and the routes between locations.

The map and itinerary allow crowd workers to see at a glance whether the plan is coherent. A worker may notice activities that are out of order, for example by seeing on the itinerary that lunch is happening too early or seeing on the map that activities can be reordered to avoid unnecessary travel. A worker can also use the itinerary to detect if too much or too little time is spent on an activity.

The itinerary doubles as an editor. Workers can drag and drop activities to rearrange their order, and click an activity to see its details, edit it, or remove it from the itinerary. On any itinerary change (i.e., via adding, removing, editing, or reordering of activities), the itinerary, activity times, map display, trip time, and todo items automatically update, which provides direct feedback to the workers as they refine the itinerary.

Mobi promotes collaboration by making the plan always visible and editable by everyone. This follows the WYSIWIS (What You See Is What I See) principle [88], which ensures that all participants have equal access to shared information. Mobi also supports *opportunistic planning*, by providing support for both top-down and bottom-up planning, and a fluid way to move back and forth between the two. For example, as workers plan at a detailed level (e.g., suggesting activities in the brainstream), they may become aware of shortcomings of the current itinerary, which in turn prompts them to start considering the itinerary as a whole. Likewise, when workers refine the itinerary, they may think of new activities to add to the brainstream, or ways to elaborate on the details of a particular activity in the current itinerary.

3.3 Experiment: Todo or Not Todo

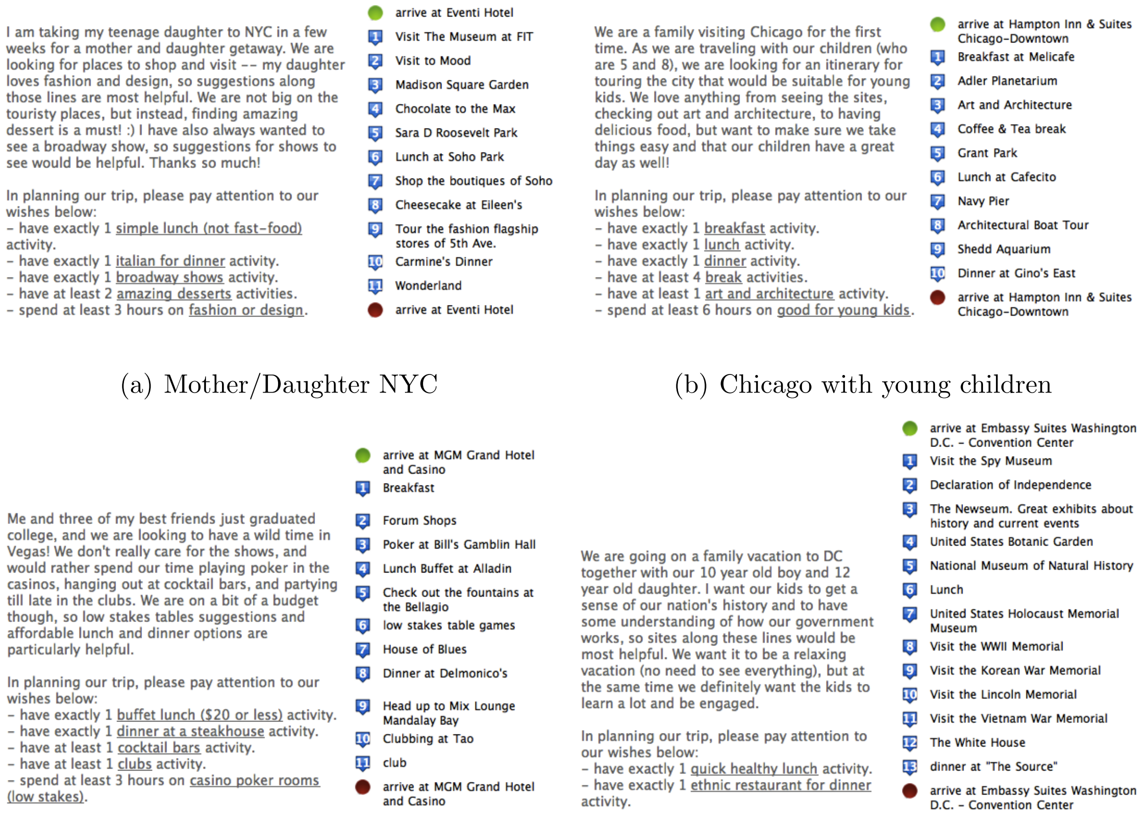
We hypothesize that elements of Mobi's design, namely the todo items and having a shared interface in which the crowd can work off the current solution context and existing ideas, promotes the crowd to effectively and collaboratively resolve the users' stated constraints so as to produce itineraries that satisfy planning missions. In this section, we consider an experiment using two versions of Mobi—one that displays todo items and one that does not—to evaluate the effect of todo items on how quickly the crowd can reach solutions that satisfy the stated quantitative constraints.

3.3.1 Method

We created custom day-trip planning missions for each of eight major U.S. cities: New York, Chicago, Washington DC, Las Vegas, Los Angeles, San Francisco, Seattle, and San Diego. We recruited Amazon Mechanical Turk workers (Turkers) in the U.S. with 95% or higher approval rating to contribute to the planning missions by working on human intelligence tasks (HITs) in which the Mobi interface was fully embedded. The interface is nearly identical to that shown in Figure 3.2, with differences being the addition of a "submit" button on the bottom, a "HIT instructions" button replacing the "what you can do to help" button in the information panel, and the addition of a "continue to improve the itinerary" todo item that displays only when there are no other todo items (all quantitative constraints are satisfied). Turkers were asked to make "micro-contributions" as they plan the trip with other Turkers, and were told that they can submit a HIT as soon as they have made *any* contribution. Turkers were paid 15 cents per HIT, and no verification was used other than requiring Turkers to have made some edit (however small) to the brainstream or itinerary before submitting the task. For half of the cities, the version with todo items was posted prior to the version without todo items, and the order of posting was reversed for the other cities. Missions were posted for up to four days. Other than the display of todo items, the interface, job description, and instructions were identical in the two conditions.

3.3.2 Results I: The Generated Itineraries

In the todo condition, all eight itineraries satisfied the stated quantitative constraints. Figure 3.5 provides four examples of planning missions and the corresponding itineraries generated by Turkers. From the itineraries, it appears that Turkers not only pay attention to the quantitative constraints, but also to the mission description, for example by including educational activities for the kids on a family vacation to Washington DC.



I am taking my teenage daughter to NYC in a few weeks for a mother and daughter getaway. We are looking for places to shop and visit -- my daughter loves fashion and design, so suggestions along those lines are most helpful. We are not big on the touristy places, but instead, finding amazing dessert is a must! :) I have also always wanted to see a broadway show, so suggestions for shows to see would be helpful. Thanks so much!

In planning our trip, please pay attention to our wishes below:
- have exactly 1 simple lunch (not fast-food) activity.
- have exactly 1 italian for dinner activity.
- have exactly 1 broadway shows activity.
- have at least 2 amazing desserts activities.
- spend at least 3 hours on fashion or design.

● arrive at Eventi Hotel
1 Visit The Museum at FIT
2 Visit to Mood
3 Madison Square Garden
4 Chocolate to the Max
5 Sara D Roosevelt Park
6 Lunch at Soho Park
7 Shop the boutiques of Soho
8 Cheesecake at Eileen's
9 Tour the fashion flagship stores of 5th Ave.
10 Carmine's Dinner
11 Wonderland
● arrive at Eventi Hotel

(a) Mother/Daughter NYC

We are a family visiting Chicago for the first time. As we are traveling with our children (who are 5 and 8), we are looking for an itinerary for touring the city that would be suitable for young kids. We love anything from seeing the sites, checking out art and architecture, to having delicious food, but want to make sure we take things easy and that our children have a great day as well!

In planning our trip, please pay attention to our wishes below:
- have exactly 1 breakfast activity.
- have exactly 1 lunch activity.
- have exactly 1 dinner activity.
- have at least 4 break activities.
- have at least 1 art and architecture activity.
- spend at least 6 hours on good for young kids.

● arrive at Hampton Inn & Suites Chicago-Downtown
1 Breakfast at Melicafe
2 Adler Planetarium
3 Art and Architecture
4 Coffee & Tea break
5 Grant Park
6 Lunch at Cafecito
7 Navy Pier
8 Architectural Boat Tour
9 Shedd Aquarium
10 Dinner at Gino's East
● arrive at Hampton Inn & Suites Chicago-Downtown

(b) Chicago with young children

Me and three of my best friends just graduated college, and we are looking to have a wild time in Vegas! We don't really care for the shows, and would rather spend our time playing poker in the casinos, hanging out at cocktail bars, and partying till late in the clubs. We are on a bit of a budget though, so low stakes tables suggestions and affordable lunch and dinner options are particularly helpful.

In planning our trip, please pay attention to our wishes below:
- have exactly 1 buffet lunch ($20 or less) activity.
- have exactly 1 dinner at a steakhouse activity.
- have at least 1 cocktail bars activity.
- have at least 1 clubs activity.
- spend at least 3 hours on casino poker rooms (low stakes).

● arrive at MGM Grand Hotel and Casino
1 Breakfast
2 Forum Shops
3 Poker at Bill's Gamblin Hall
4 Lunch Buffet at Alladin
5 Check out the fountains at the Bellagio
6 low stakes table games
7 House of Blues
8 Dinner at Delmonico's
9 Head up to Mix Lounge Mandalay Bay
10 Clubbing at Tao
11 club
● arrive at MGM Grand Hotel and Casino

(c) Vegas with buddies

We are going on a family vacation to DC together with our 10 year old boy and 12 year old daughter. I want our kids to get a sense of our nation's history and to have some understanding of how our government works, so sites along these lines would be most helpful. We want it to be a relaxing vacation (no need to see everything), but at the same time we definitely want the kids to learn a lot and be engaged.

In planning our trip, please pay attention to our wishes below:
- have exactly 1 quick healthy lunch activity.
- have exactly 1 ethnic restaurant for dinner activity.

● arrive at Embassy Suites Washington D.C. - Convention Center
1 Visit the Spy Museum
2 Declaration of Independence
3 The Newseum. Great exhibits about history and current events
4 United States Botanic Garden
5 National Museum of Natural History
6 Lunch
7 United States Holocaust Memorial Museum
8 Visit the WWII Memorial
9 Visit the Korean War Memorial
10 Visit the Lincoln Memorial
11 Visit the Vietnam War Memorial
12 The White House
13 dinner at "The Source"
● arrive at Embassy Suites Washington D.C. - Convention Center

(d) Family in DC

Figure 3.5: Experiment: planning missions and the corresponding itineraries generated by the crowd in the todo items condition

Table 3.1 summarizes, for each of the examples shown in Figure 3.5, statistics about the final itineraries, the types of edits Turkers made, and the amount of money paid to workers. We see that the final itineraries contain original ideas from multiple workers. Turkers generated just over twice as many ideas for activities as are in the final itineraries, and generally used notes sparingly. When notes were added, they provided commentary on alternative suggestions ("They are a better place then Pasty's by far, and have better service, plus that perfect dessert."), noted errors in activities ("Barbary Coast isn't called 'Barbary Coast' anymore"), presented general

	NYC	Chicago	Las Vegas	DC
# unique workers	17	15	16	21
# workers with winning ideas	6	5	7	8
# activities in brainstream	35	16	18	28
# activities in itinerary	11	10	11	13
# edits in brainstream	50	54	43	57
# edits in itinerary	193	140	75	154
# notes in brainstream	1	0	9	1
# of HITs	64	31	47	50
Total cost	$9.60	$4.65	$7.05	$7.50

Table 3.1: Summary statistics about the final itineraries of the examples shown in Figure 3.5, including contributions by and payments to Turkers. Winning ideas are activity suggestions that are in the final itinerary.

advice ("You can buy a MealTicket which will allow you to eat free at many places."), or pointed out problems with the plan ("why are we eating so much dinner?").

3.3.3 Results II: Impact of Todo Items

Results show that when prompting workers with todo items, quantitative constraints are satisfied significantly more quickly than when todo items are not displayed. We measure the speed at which constraints are satisfied in number of HITs performed. One worker in the no todo condition attempted to game the system by submitting multiple HITs for a single piece of work (e.g., by adding an activity, filling in its details, and placing it into the itinerary in three separate HITs). For this worker, only the itinerary-changing HITs were counted, but for all other workers, all HITs were counted.

We make three observations. First, we found a significant difference ($t(7) = 3.65$, $p = 0.0082$) in the number of HITs it took to satisfy (for the first time) all of the stated quantitative constraints between the todo condition ($\mu = 16.5$, $\sigma = 9.65$) and the no

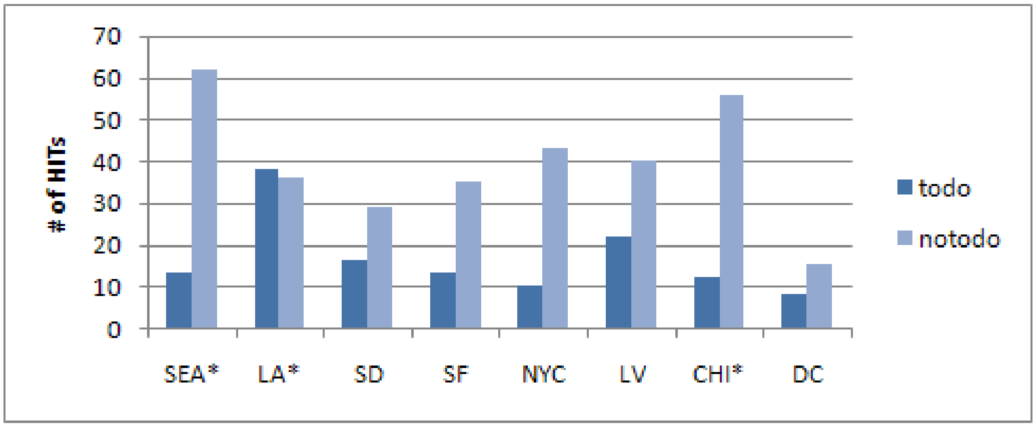


Figure 3.6: Number of HITs required to satisfy all quantitative and system generated time constraints for each city in the todo and no todo conditions. For cities marked by an asterisk, itineraries in the no todo condition still had violated constraints; in such cases we reported the number of HITs thus far.

todo condition ($\mu = 39.5$, $\sigma = 14.8$).[2] See Figure 3.6 for a city-by-city breakdown.

Second, there is also a significant difference ($t(7) = 4.247$, $p = 0.0038$) in the number of HITs it took to satisfy all constraints for the first time (this includes system generated time constraints) between the todo condition ($\mu = 22.5$, $\sigma = 8.5$) and the no todo condition ($\mu = 45.38$, $\sigma = 13.9$).

Finally, as constraints can be violated and satisfied repeatedly throughout the planning process, we sought to understand how quickly constraints are satisfied on average. We introduce the notion of the *violation duration* of a constraint, which is the number of HITs it takes for a constraint to be satisfied by the itinerary since it was last violated (which could be when it was first introduced). The average violation duration of quantitative constraints is shorter for the todo condition ($\mu = 5.64$, $\sigma = 6.34$) than for the no todo condition ($\mu = 10.5$, $\sigma = 10.97$); the result is statistically significant

[2] In some cases for the no todo condition, no itinerary satisfied all the stated requirements in the course of the experiment. In such cases the number of HITs completed thus far was used as a lower bound for comparison.

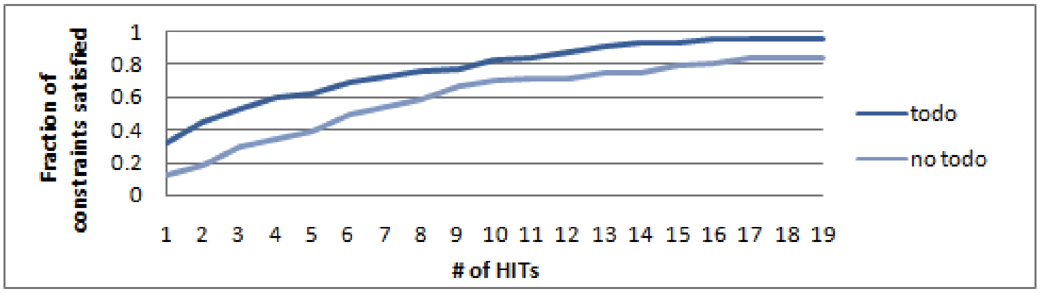
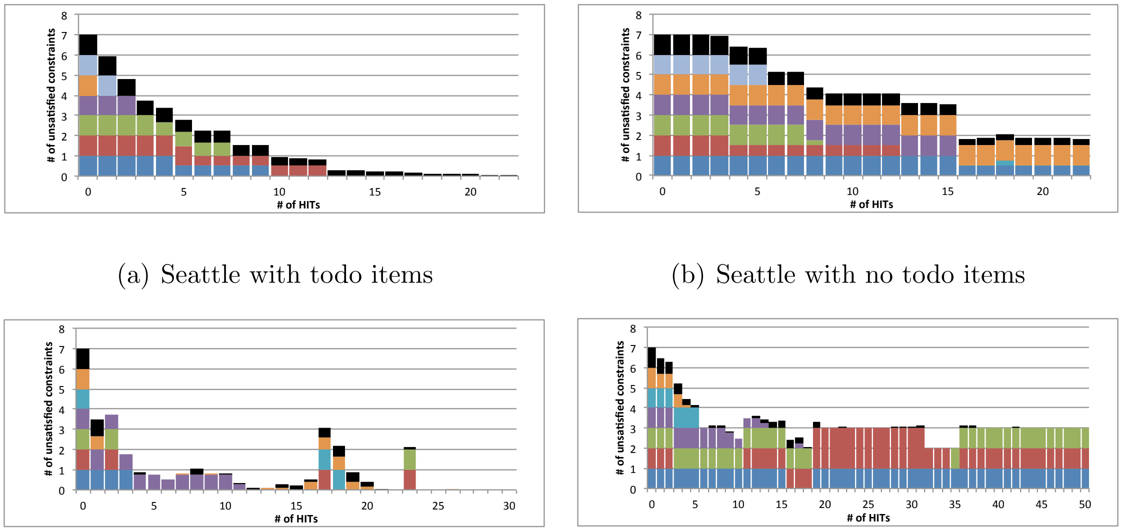


Figure 3.7: Cumulative distribution of the violation duration of constraints in the todo versus no todo conditions, showing the fraction of constraints satisfied after at most k HITs since the time it was last violated.

$(t(134) = 3.206, p = 0.0017)$.

Figure 3.7 shows the cumulative distribution of the violation durations of constraints in the todo versus no todo conditions. We observe that for any violation duration we may consider (in number of HITs), a larger fraction of the constraints are satisfied within that duration in the todo condition than the no todo condition. We also see that more than half of all violated constraints were satisfied after three or fewer HITs in the todo condition.

Figure 3.8 shows, for the todo versus no todo conditions, the rate at which each constraint gets satisfied as workers contribute to the planning effort for the Seattle and Chicago planning missions. We observe that constraints were satisfied much more quickly in the todo condition. The Chicago case is particularly interesting. In the todo condition, a worker violated a previously satisfied constraint while editing and proceeded to make successive edits that led to the satisfaction of all constraints. In the no todo condition, a satisfied constraint was violated and then left unaddressed. This example illustrates the power of immediate feedback. When an edit to the itinerary violates some constraint(s), the automatically generated todo items are able to not



(a) Seattle with todo items

(b) Seattle with no todo items

(c) Chicago with todo items

(d) Chicago with no todo items

Figure 3.8: The unsatisfied constraints during the planning process for the Seattle and Chicago missions. The height of each bar indicates the number of constraints unsatisfied after k HITs. Each colored segment represents a particular quantitative constraint, and its height indicates the extent to which it is violated. The black segment represents the percent by which the itinerary is over time or under time (when it is greater or less than 5%).

only alert workers to what needs fixing, but also make them aware that their edits have a direct effect on the satisfaction of constraints associated with the planning mission.

3.3.4 Results III: Editing Patterns

Having shown that todo items play an important role in focusing the crowd's effort towards satisfying the quantitative constraints, we turn to investigate the crowd's work process while using Mobi in the todo condition. In particular, we look for evidence of collaborative behavior from the crowd and examine the way that they plan using the current context of the plan.

We focus first on the process of generating ideas for activities. We observe that roughly half (52%) of the contributions to the brainstream contain new suggestions while the other half (48%) are edits to existing ideas in the itinerary. Of the edits, 72% are edits on ideas that originated from someone other than the person editing, which suggests that workers are working off others' contributions when they refine ideas and the itinerary. When editing an activity, we see that edits are mostly to an activity's duration (80%), but there are also edits to change titles/descriptions (7%) and to correct an activity's location (12%). Edits to the title, description, and location are encouraging to see as they suggest that the brainstream and itinerary viewer are providing means for users to discover and improve existing ideas.

Turning to the patterns of itinerary edits, we observe that while most of the contributions come from adding (31%) and reordering activities (32%), workers also edit existing ideas (22%) and remove activities (14%). This is encouraging to see because workers are using the different actions available to them to improve the itinerary as they see fit. When tasks are left to run after the quantitative constraints are all satisfied, we observe that itineraries continue to evolve; workers replace activities in the itinerary with other activities, reorder the itinerary, edit existing items, and so on. While constraints may be violated during such edits, todo items reminded workers of such violations and violated constraints were quickly satisfied again (e.g., see Figure 3.8(c)). Workers are encouraged to continuously generate new ideas and incorporate them into the itinerary both because we pay them for such contributions and because Mobi displays a todo item that asks workers to continue improving the itinerary whenever all quantitative constraints are met.

We saw very few Turkers who blatantly tried to game the system. The kinds of gaming behavior we did observe generally fell into two categories. In one, a Turker underspecifies an activity, either by creating an activity without filling in its description and location, or by adding a note containing a suggestion for an activity instead of just adding the suggested activity. In the other, a Turker would fully specify an activity, but use up to three HITs to do so—by spending a HIT on creating the activity, another to edit its details, and another to add it to the itinerary—when all this can be accomplished with a single “add activity” action.

While it is certainly useful to consider refinements that would curb such behaviors (e.g., by requiring activities to contain descriptions; by not allowing workers to submit HITs in which they have only edited their own ideas; etc.), such gaming behaviors from a few Turkers did not seem to have a negative influence on the planning processes nor the resulting solutions. In particular, we saw that poorly formed ideas were simply ignored, removed from the itinerary, or edited by another worker who discovered them via the autocomplete search box in the brainstream, all of which occurred as natural parts of the iterative process through which workers improved the itinerary.

3.4 End-to-End User Study

Having seen that workers can resolve quantitative constraints effectively using Mobi, we conducted a user study to evaluate how well the generated itineraries satisfy not only quantitative constraints, but also the stated qualitative constraints, from the perspective of requesters.

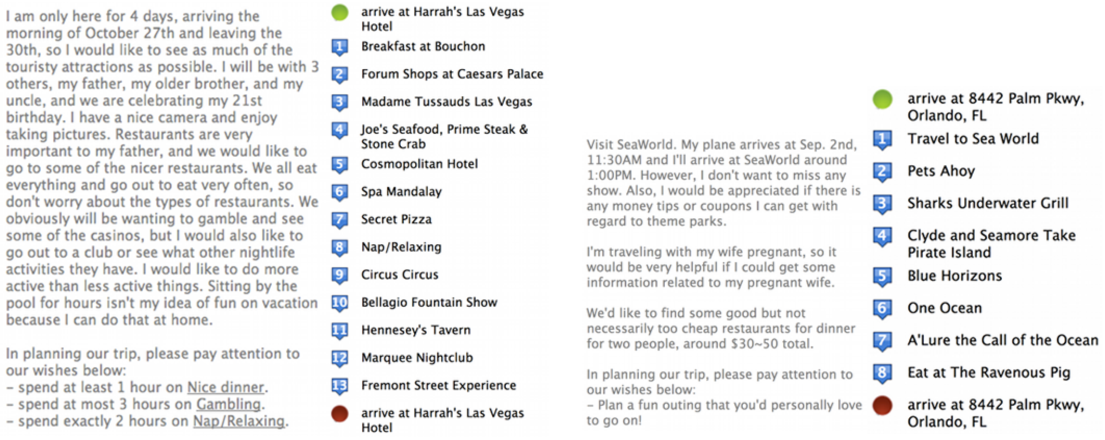
3.4.1 Method

We recruited 10 subjects from university mailing lists to participate in the study. Individuals were eligible if they were actually planning a forthcoming trip to a major U.S. city. Recruited subjects were a mix of undergraduate students, graduate students, and research scientists. Subjects were instructed to describe their planning mission, which includes qualitative and quantitative preferences and constraints. Participants were given unlimited access to Mobi for a week, during which they were free to modify their planning mission and participate in the planning process. Missions were crowdsourced on Mechanical Turk as was done in the todo versus no todo experiment. At the end of the study, subjects completed a questionnaire, which asked them to evaluate the final itinerary and to describe their experience using Mobi. Subjects each received a $30 Amazon Gift Card for their participation.

The trip destinations specified by the users included Boston, New York City, San Francisco, Las Vegas, Orlando, and Washington DC. The planning missions varied in length and specificity. Figure 3.9 provides two examples of user missions and the generated itineraries.

3.4.2 Results

To assess how well the generated itineraries satisfy the users' requirements, we consider three measures of the *quality* of an itinerary, namely the extent to which it (1) contains activities that the requester likes, (2) satisfies the qualitative and quantitative constraints specified in the planning mission, and (3) serves its purpose as a plan that is feasible, useful, and executable in real life.



I am only here for 4 days, arriving the morning of October 27th and leaving the 30th, so I would like to see as much of the touristy attractions as possible. I will be with 3 others, my father, my older brother, and my uncle, and we are celebrating my 21st birthday. I have a nice camera and enjoy taking pictures. Restaurants are very important to my father, and we would like to go to some of the nicer restaurants. We all eat everything and go out to eat very often, so don't worry about the types of restaurants. We obviously will be wanting to gamble and see some of the casinos, but I would also like to go out to a club or see what other nightlife activities they have. I would like to do more active than less active things. Sitting by the pool for hours isn't my idea of fun on vacation because I can do that at home.

In planning our trip, please pay attention to our wishes below:
– spend at least 1 hour on Nice dinner.
– spend at most 3 hours on Gambling.
– spend exactly 2 hours on Nap/Relaxing.

● arrive at Harrah's Las Vegas Hotel
1 Breakfast at Bouchon
2 Forum Shops at Caesars Palace
3 Madame Tussauds Las Vegas
4 Joe's Seafood, Prime Steak & Stone Crab
5 Cosmopolitan Hotel
6 Spa Mandalay
7 Secret Pizza
8 Nap/Relaxing
9 Circus Circus
10 Bellagio Fountain Show
11 Hennesey's Tavern
12 Marquee Nightclub
13 Fremont Street Experience
● arrive at Harrah's Las Vegas Hotel

(a) Subject 1: Las Vegas

Visit SeaWorld. My plane arrives at Sep. 2nd, 11:30AM and I'll arrive at SeaWorld around 1:00PM. However, I don't want to miss any show. Also, I would be appreciated if there is any money tips or coupons I can get with regard to theme parks.

I'm traveling with my wife pregnant, so it would be very helpful if I could get some information related to my pregnant wife.

We'd like to find some good but not necessarily too cheap restaurants for dinner for two people, around $30~50 total.

In planning our trip, please pay attention to our wishes below:
– Plan a fun outing that you'd personally love to go on!

● arrive at 8442 Palm Pkwy, Orlando, FL
1 Travel to Sea World
2 Pets Ahoy
3 Sharks Underwater Grill
4 Clyde and Seamore Take Pirate Island
5 Blue Horizons
6 One Ocean
7 A'Lure the Call of the Ocean
8 Eat at The Ravenous Pig
● arrive at 8442 Palm Pkwy, Orlando, FL

(b) Subject 2: Orlando

Figure 3.9: User study: planning missions and corresponding itineraries generated by the crowd

1. *Do itineraries contain activities that the requesters like?*

Users were shown information about each of the itinerary activities (title, description, start time, end time, duration) and asked to rate how much they think they would enjoy each activity on a 5-point scale (1="hate it", 5="love it").

Figure 3.10 shows a histogram of the activity ratings across all 10 participants. The mean rating was 4.03 ($\sigma = 0.44$). Users also mentioned that the activities are diverse, interesting, and often unknown to them prior to using Mobi.

2. *Do itineraries satisfy the qualitative and quantitative constraints specified in the planning mission?*

All of the users answered that their itinerary fulfilled most or all of the requirements they had specified. Some users noted specific activities that they did not like, such as one who commented "I just happen to be afraid of bungee jumping because it seems so unsafe, but a similar activity would be fun" and another who commented "I am under age so the wine thing would not be great for me but everything else

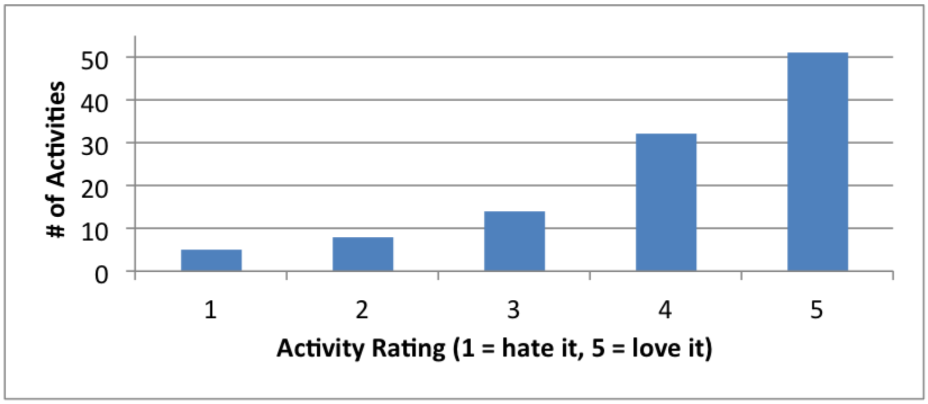


Figure 3.10: Histogram of activity ratings

sounds great." Another user complained about too many activities: "There was far far far too much packed into a single day, but the ideas were all totally interesting." Yet another user felt they were visiting too many parts of a city in one day: "For the most part, it was a good mix of things to do. I was not expecting to travel so much uptown/downtown in one day though."

These problems can be explained in part by the fact that some constraints, such as the notion that an itinerary shouldn't be too packed, are assumed or missed and therefore not explicitly stated by the users. One potential solution is for requesters to evaluate the itineraries as they are being created and add the missing constraints to the planning mission. In fact, as a preliminary test, we took two users' feedback and entered them as todo items (i.e., "Let's just stay midtown and remove downtown activities," "The harbor island suggestions are great but one island would be enough. Please adjust time durations accordingly so the day is not so packed."). We observe that after just a few HITs, workers have already addressed the issue by removing offending activities, reordering activities (so that meals occur at reasonably hours), and adding additional activities (replacing a concert downtown with a Broadway show in midtown).

3. Are the itineraries feasible, useful, and executable in real-life settings?

We asked users if they *would* or *did* use the itinerary in real-life. All users expressed that they would use the itinerary as is, some version of the itinerary, or selected ideas from the itinerary. When asked *"If Mobi were made available for general use, how likely would you want to use such a tool again for recruiting the crowd to help you plan a trip?"*, 7 out of 10 users answered *likely* or *very likely*, 2 answered *neutral* and only 1 answered *unlikely*.

Three users actually followed the itinerary or used the ideas in the itinerary in their real-life trips. One user reported that "having other people involved in the idea-creation process was extremely helpful. It sparked all sorts of ideas that I kept in the back of my head throughout the weekend." Another user remarked that his "trip was mostly in the plan," although his restaurant plans changed during the trip.

We found a dichotomy of users: those who are interested in obtaining a *fully-specified itinerary* and those who are interested in a *loose itinerary* that contains an unordered set of suggested activities that leaves room for exploration. A possible solution is to allow requesters to choose between a fully specified or loose itinerary, which in turn translate into constraints that specify the maximum number of activities in the itinerary, the amount of buffer time between activities, and the extent to which activities need to be ordered.

One of the most frequently mentioned benefits of Mobi is that both the *idea generation* and the *planning* are fully automated, thereby "integrating all the factors one would consider in planning an itinerary," yet making "the time spent creating the plan minimal." Most users (7 out of 10) reported that they were comfortable with an anonymous crowd planning their trip. Furthermore, results show that requesters mostly left the planning up to the crowd. In particular, 3 out of 10 users reported that they never or rarely checked on the progress of the itinerary, 5 did so occasionally, and only 2 did so frequently. Likewise, 7 out of 10 users said that they never went back to modify the mission details or add notes. As one user noted: "the process seemed to work smoothly without my intervention."

3.5 Discussion

Having demonstrated the effectiveness of Mobi for helping the crowd to resolve qualitative and quantitative constraints in the itinerary planning setting, we now revisit the elements of Mobi's design and discuss how these elements may in general inform the design of systems that facilitate a crowd to tackle problems involving global constraints.

Keeping the crowd, the solution, and the context together

Compared to the design of most other crowdsourcing systems for tackling complex tasks, Mobi is distinguished in its use of a single structured interface through which the crowd is exposed to the current solution and the global problem-solving context. This unified view provides a shared context that allows contributors to coordinate and communicate more effectively with one another than approaches where participants work on different subtasks in separate contexts.

Interactions are less controlled, but still structured

Mobi allows workers to choose how they want to contribute to the task. In our studies, we found that workers generate diverse sets of ideas, and make various types of contributions while problem solving. This freedom is particularly important for resolving global constraints as we do not know *a priori* the specific contributions that are needed. Rather, contributions are context dependent. While interactions are less controlled this way, they are still highly structured. The crowd selects from a well-specified set of actions, todo items guide the crowd towards useful actions, and the system provides real-time feedback on the effects of actions.

A language for human-computer communication

In the background, Mobi's automation computes routes and times, checks for violated constraints, and generates todo items. Mobi understands, for example, when all of the quantitative constraints are satisfied. This ability enables Mobi to take actions such as prompting the crowd for future revisions and asking the crowd or requester to check for potential problems. Mobi can do these things without knowing what the constraints mean, because the inputs that it seeks from the crowd include the categories of suggested activities. This information is sufficient for the system to check for violated constraints and therefore assist in the planning process.

A fluid way to refine goals

With complex problems, requirements can change over time as ideas and partial solutions stream in. In Mobi, a requester can add or revise requirements, write notes, or even directly alter the plan during the planning process. The crowd can react to such changes just as they react to the current solution at any other point in the planning process. The iterative nature of the task and the ease with which workers can grasp the current solution and access alternative suggestions make it easy for the crowd to see and respond to such changes.

3.6 Summary and Research Directions

To date, many human computation systems have relied on the assumption that problems can be solved in an algorithmic manner, using explicit procedures that outline the operations that need to be done and how they are ordered. In this chapter, we argue for an alternative *crowdware* approach, in which workers contribute to solving a complex problem in a less controlled environment that allows them to view and build upon each other's ideas and to contribute as they wish, while system-generated alerts and advice guide them towards a solution.

Using itinerary planning as a case study, we introduce Mobi, a system that draws on groupware ideas and uses explicit processes such as the automatic generation of todo items to generate itineraries that satisfy complex, interdependent constraints. Our results show that constraints are resolved efficiently using this design, and that end user found that the generated itineraries satisfied their stated quantitative and qualitative constraints.

On Mobi, we are interested in studying ways to handle the implied constraints that are assumed or missed. The challenge is to make implied constraints visible so they can be tackled like other constraints; possible approaches include having the crowd identify them, using automated procedures to detect and learn about such constraints, and asking requesters to provide feedback. In a related direction, we can also attempt to encapsulate qualitative constraints in todo items, which would allow workers to see everything that needs work in one place. In addition, we envision rich opportunities to integrate different types of automation into Mobi—to detect failures, handle uncertainties, incorporate richer forms of user preferences, and combine automated and human planners in a synergistic way.

On crowdware more generally, we are interested in understanding how to focus the crowd's attention on not only resolving global constraints, but on taking actions that are most likely to lead to high quality solutions. This may involve generalized uses of todo items to focus the crowd's attention and effort on where they are mostly likely to matter, for example by taking into account the potential value of possible refinements. Such value judgments may be guided by heuristic evaluations made by the requester, the crowd, or the automated system. Focusing the crowd on solution quality may also involve taking steps to ensure that a solution is coherent and consistent, and is not suffering from issues often associated with "design by committee." Steps may include explicitly promoting consistency checks, engaging the crowd in making high-level judgments about the coherence of the solution, and having the crowd and automated system make decisions about when to seek feedback from the requester.

In addition to these directions, there are opportunities to explore novel combinations of crowdware and workflow approaches that can enhance the ability of participants to effectively contribute to solving complex problems that are hard to decompose. We elaborate on this direction in Chapter 9.

Chapter 4

Harnessing Crowd Abilities: Control and Synthesis

Human computation algorithms tend to define an explicit sequence of steps in which individuals in the crowd are recruited to complete subroutines within this predefined process. But in the previous chapter we introduced Mobi, a system that allows the crowd to shape the problem-solving process directly by contributing opportunistically while being guided by system-generated alerts. In this chapter, we develop a broader perspective on how the crowd can contribute to problem-solving efforts, by considering opportunities for the crowd to guide the control flow of an algorithm and generate plans that define the problem-solving process.

From a computational perspective, we envision that individuals in a crowd can play diverse roles in an organized problem-solving process. People can not only serve as data oracles at the endpoints of computation, but also as modules for decomposing problems, controlling the algorithmic progression, and even generating plans and synthesizing programs for solving problems. From an organizational perspective, individuals in the crowd may take on roles beyond "doing the work"—including defining and communicating subgoals, evaluating the value of current solutions, and routing tasks to appropriate individuals. The crowd may also be made aware of time or other resource constraints, and be asked to make tradeoffs between further deliberation versus taking time-critical actions.

In exploring new ways in which the crowd can contribute to problem solving, we aim to derive principles and methods for *crowdsourcing general computation*, that can enable general problem solving via human computation systems. By drawing on the *general intelligence* of the crowd, we can enable the crowd to tackle more creative, open-ended tasks, while also bringing about more effective and efficient problem-solving processes. On the one hand, by contributing diverse knowledge, expertise, and sensing capabilities, the crowd can potentially tackle complex problems that are difficult for individuals. On the other hand, as in our study of human computation algorithms and crowdware, individuals in the crowd may only be briefly involved and may contribute noisy solutions. Extending the crowd's problem-solving abilities to control, synthesis, and beyond will likewise have to account for limitations of the crowd, and provide mechanisms to support effective coordination.

Section 4.1 reviews related work in crowdsourcing and artificial intelligence. Section 4.2 describes various ways the crowd may guide the control flow of an algorithm. Focusing on the 8-puzzle as an illustrative example, we show how by passing context a crowd can solve difficult problem instances that the crowd struggles on when not passing context. Section 4.3 explores using the crowd as a general purpose planner.

We present CrowdPlan, a system that takes a high-level problem in natural language as input and recruits a crowd to break down the problem into a simple plan, resulting in a novel form of interaction for Web search. Section 4.4 closes the chapter with a summary of results and discussion of research directions.

4.1 Related Work

In addition to CrowdPlan and Mobi, a number of recent human computation systems have started to take advantage of the crowd's ability to plan and execute solutions. Boujarwah et al. [8] introduced a system for crowdsourcing social scripts that consist of steps, obstacles, and solutions to complex social scenarios, which are used to support social problem-solving skills for individuals with autism. Kokkalis et al. [49] introduced TaskGenies, a crowd-powered task management system that provides action plans to help and encourage users to complete tasks. Kulkarni et al. [50] introduced Turkomatic, a system that involves the crowd in concurrently planning and executing plans for solving complex tasks. To synthesize a plan, Turkomatic involves the crowd in making control decisions, by deciding whether to solve problems directly or to decompose them further. To ensure that worker-generated plans are feasible, Turkomatic also allows requesters to intervene and guide the planning and execution, suggesting interactions in which both the crowd and the requester contribute to general problem solving.

Analogous to our study of general problem solving with crowds, the field of artificial intelligence also concerns itself with general problem solving, but from the perspective of machine agents. Studies of metareasoning [36, 80] aim to design agents that can not only reason about specific problems, but also make decisions about what to reason about, how long to deliberate, and when to take actions. Adopting the view that we only have bounded time and computational resources available, metareasoning procedures aim to make more efficient use of resources for deliberation and action through higher-level reasoning about the problem-solving process [37, 9]. Principles and techniques for metareasoning may provide an interesting perspective for the design of metareasoning procedures for crowds, and may also be used more directly to automatically control human computation processes or synthesize workflows. This latter perspective is explored in more detail in Chapter 8.

4.2 Crowd as Controllers

In the process of problem solving, humans may have useful intuitions about how best to proceed based on the current solution context. Below we describe a few promising directions for engaging the crowd to guide the control flow of an algorithm:

- **Decompose versus solve**

  In Chapter 2, we introduced divide-and-conquer as a useful design pattern for decomposing a problem into subproblems, and for composing solutions of subproblems into a solution. For open-ended tasks in which the crowd performs the decomposition, the difficulty of resulting subtasks may be hard to determine a priori. Instead of predetermining how much a problem should be decomposed before requesting a solution to a subproblem, it may be helpful to give the crowd the option to either solve a problem completely, or to decompose the problem for the crowd to then solve or decompose further. The crowd's decisions would make implicit tradeoffs between the costs of different stages of computation, potentially enabling more efficient problem solving while also allowing individuals to make decisions based on how much effort they are willing and able to contribute. For example, Zhang et al. [105] introduced a system called TurkSort, which crowdsourced tasks from a quicksort algorithm to Mechanical Turk workers who contributed by finding pivots, partitioning, or sorting, at their choosing. By giving workers the choice of sorting the current list or decomposing the list further, the base case of the recursion was defined implicitly by workers' decisions. As another example, when synthesizing a workflow for solving a problem, Turkomatic [50] workers were asked to judge whether the current price for a task is fair, and if not to decompose it into simpler tasks, with this process repeated recursively.

- **Transmitting solution context and subgoals**

  As part of problem solving, some computational methods track and pass parameters on local and global states and on measures of progress. Human computation may face similar challenges with sharing context among workers about problem-solving strategy and state, particularly when the computation is divided into small pieces performed by many workers. Unless a decomposition is defined or context about what work remains is shared, it may be hard for people to contribute effectively. For example, in the Mobi experiment in Section 3.3, we showed that the absence of todo items significantly increased the amount of time taken to resolve constraints. While we can sometimes rely on the system to provide the necessary context, we can also engage the crowd in sharing solution context and subgoals. Such actions may enable more efficient problem solving, by helping subsequent contributors to make better decisions and allowing good problem-solving strategies to be passed forward.

- **Controlling search processes**

  Tasks like itinerary planning can be viewed as search problems, in which the crowd is iterating on the current itinerary in search for an effective plan from a large space of possible plans. In this and other search problems, the ability to guide the search process towards good neighborhoods and to backtrack when necessary are important components of an effective search method. With a human computation approach to these problems, people can assess the current solution state, decide which neighborhood(s) to search in, and backtrack when further improvements from the current state are unlikely.

4.2.1 Case Study: 8-Puzzle

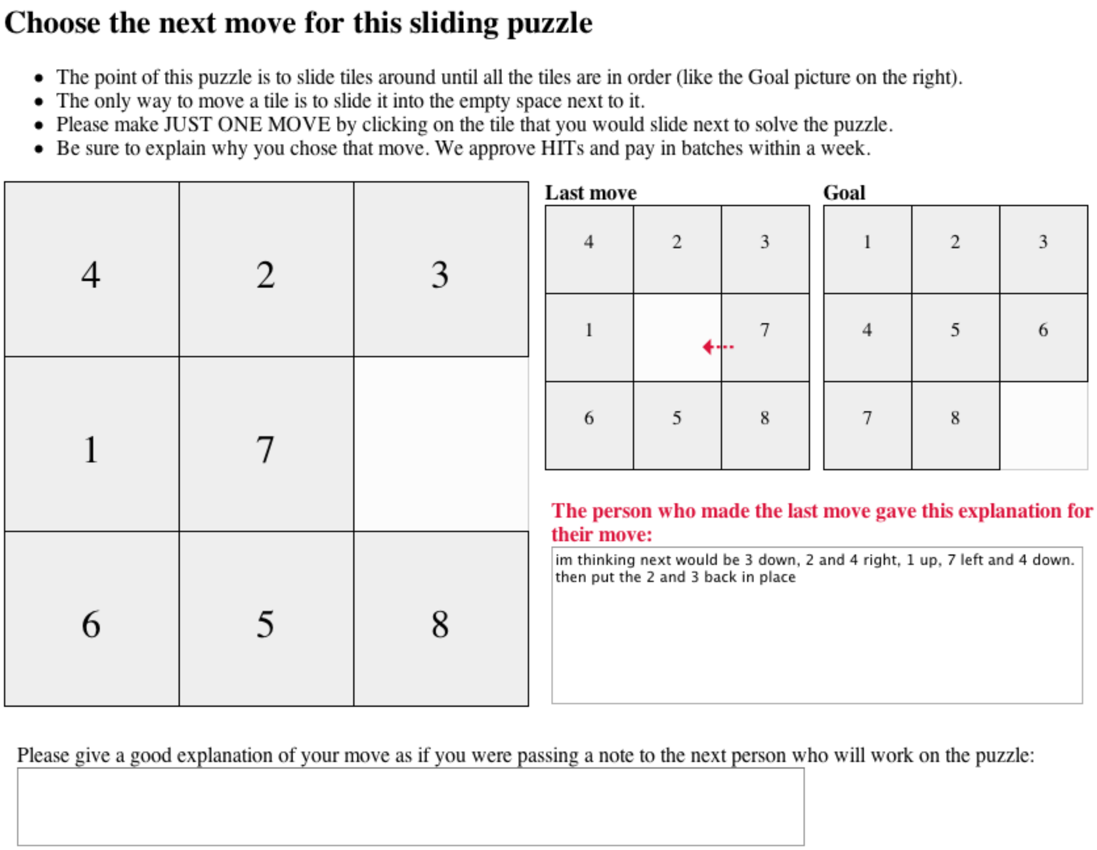
To illustrate how engaging the crowd in control can lead to more effective problem solving, we present a study of the 8-puzzle. In the 8-puzzle, a 3x3 board holds eight tiles numbered from 1 through 8. The goal is to slide tiles on the board until the numbers on the tiles are in numerical order. To understand how workers may deal with limited problem-solving context, we allow each worker to make just one move. This simple setting serves as a model for more complex problems we may wish to crowdsource, like writing an article or a piece of code, where a crowd contributes iteratively with each worker expected to make only a small contribution.

In the 8-puzzle example, a worker needs to know enough about what they should work on to make effective progress on a subgoal at hand, and know how the subgoal fits within the overall aim. Given limited context, workers may get stuck on difficult board positions. Thrashing can occur with successive contributions revisiting the same states. One can imagine allowing workers to discuss strategies and pass the entire discussion from worker to worker, but if the cost of understanding the context dominates the time that a worker is willing to contribute, this kind of collaboration may become costly, ineffective, or even impossible.

We seek to understand whether it is possible to pass along a small amount of context from worker to worker—with no formal agreements on subgoals—while still making progress towards the goal. To do this, we designed a task in which each worker is provided with the last person's move and their short explanation for making that move. The worker is asked to decide on the next move, and similarly to provide a short explanation for their move. Figure 4.1 shows the workers' task interface.

In an experiment, we compared the performance of the crowd on this task with a version of the task in which workers were only provided with the current board position and not the last worker's move and explanation. Instructions for the two settings are otherwise identical. We recruited workers on Mechanical Turk (Turkers), who were each paid 5 cents per move. To prevent the same worker from making consecutive moves and dominating the problem solving, we only allowed a worker to return to a particular puzzle after five moves have been made by other workers.

We consider 20 problem instances, divided evenly into "medium" and "hard" difficulty, as determined by the minimum number of steps required to reach the goal



Choose the next move for this sliding puzzle

- The point of this puzzle is to slide tiles around until all the tiles are in order (like the Goal picture on the right).
- The only way to move a tile is to slide it into the empty space next to it.
- Please make JUST ONE MOVE by clicking on the tile that you would slide next to solve the puzzle.
- Be sure to explain why you chose that move. We approve HITs and pay in batches within a week.

4	2	3
1	7	
6	5	8

Last move

4	2	3
1	←	7
6	5	8

Goal

1	2	3
4	5	6
7	8	

The person who made the last move gave this explanation for their move:

im thinking next would be 3 down, 2 and 4 right, 1 up, 7 left and 4 down. then put the 2 and 3 back in place

Please give a good explanation of your move as if you were passing a note to the next person who will work on the puzzle:

Figure 4.1: Task interface for an iterative step in the 8-puzzle game, where each Turker is shown the last Turker's move and explanation for that move. Here the previous Turker moved the 7 tile and recommended the next sequence of moves.

configuration from the initial board configuration. Medium instances required between 12 to 16 steps, while hard instances required between 22 to 26 steps. We allowed each instance to run until the puzzle was solved or for at most 100 steps.

Our results show that in the condition with context passing, all puzzles were solved before 100 steps were reached. In the condition without context passing, 9 of the 10 puzzles were solved for medium difficulty puzzles, and only 5 of the 10 puzzles were solved for hard difficulty puzzles. In addition to completing more puzzles, context passing also reduced the number of iterations Turkers took to complete puzzles. Considering all instances, a Wilcoxon test shows a significant difference (z = -2.61, $p < 0.01$) between the number of steps before a puzzle is solved (or stopped after 100 steps) in the context passing condition ($\mu = 38.6$) and the no context passing baseline ($\mu = 55.9$).

In looking through the problem-solving process, it is apparent that communication can be very useful in some instances. See Figure 4.1, where a previous Turker identified a path forward and noted it for the next Turker. Had he not contributed that action and highlighted the path, the problem would likely have been more difficult to solve and more steps would have been taken. The ability to pass on context gives the next player a better idea of how to proceed, raising the probability that progress will be made toward the solution. We also observe that Turkers sometimes passed on the advice from previous players that they deemed useful, while at other times suggested alternative moves and directions when they found suggestions unhelpful.

4.3 Towards Human Program Synthesis

As the crowd engages in algorithmic control, humans are no longer limited to providing outputs for predefined modules, but can fill in parameters of the algorithm itself and make evaluative decisions to define the best path through a solution space. An interesting question is whether a crowd can go beyond algorithm control towards the notion of *synthesis*. In machine computation, *program synthesis* considers the use of appropriate design tactics to systematically derive a program based on a problem specification. For example, the synthesis of a divide-and-conquer algorithm [86] may involve the derivation of a tree of specifications, where leaves in the tree represent subproblems for which solutions can be readily provided, and instructions for recomposition are also derived. Taking the analogy to the crowd, we can seek to enlist a crowd in both program synthesis and program execution. By considering problems that the crowd is well-suited for, we can engage the crowd to construct an overall plan for the problem-solving process and to execute the plan. Such plans can include decomposing a problem into subproblems, solving the subproblems, and then recomposing solutions to subproblems into a solution.

4.3.1 Case Study: Collaborative Planning for Web Search

As a first step towards program synthesis with a crowd, we consider an application to Web search. Web search is a difficult AI problem. To date, research on Web search has focused primarily on improving the relevance of search results to a query. However, people use the Web not only to retrieve relevant information, but to solve short-term or long-term problems that arise in their everyday lives. While current search engines are able to provide relevant information in response to well-specified queries, the heavy burden of actually solving a problem (e.g., figuring out what steps to take, how to accomplish these steps, and what queries to enter to find helpful resources) is placed entirely on the user. For a user with a *mission* in mind, e.g., “I want to get out more,” or “I need to manage my inbox better,” a typical search scenario today would involve the user digging through a set of blogs, opinion or “how-to” articles on the Web in order to identify important subproblems, and then submitting multiple search queries to find resources for addressing each subproblem.

We envision the next generation of search engines to more closely resemble interactive planning systems. They would be able to take in high-level mission statements ("I want to ...") as input and directly generate plans to achieve these missions. For example, a simple plan may detail specific steps to take, provide explanations for why these steps are important, and return relevant resources for accomplishing each step. A more complex plan may even include conditional branches and recourse decisions, for example to handle situations when a step does not work as intended.

Unfortunately, the gap between the capabilities of current search engines and the envisioned next-generation search engines is huge. A system would have to not only understand natural language missions, but also be equipped with large amounts of common-sense and real-world knowledge about solving problems of interest.

To fill this gap, we introduce *CrowdPlan*, a human computation algorithm that takes a high-level mission as input and returns a simple plan that captures the important aspects of the user's problem as output. CrowdPlan leverages human intelligence to decompose a *mission* into low-level *goals*, which are then mapped into queries and passed onto existing search engines.[1] The output is a simple plan consisting of a set of goals for tackling different aspects of the mission, along with search results tailored to each goal. For example, the high-level mission "I want to live a more healthy life" can be decomposed into a variety of goals, including "stop smoking," "eat healthier foods," "exercise," "drink less alcohol," "spend time with family," and "sleep more." Each of these goals, in turn, can be supported by one or more search queries. For

[1] We adopt the definitions in Jones and Klinkner [42], and define a *goal* as "an atomic information need, resulting in one or more queries" and a *mission* as "a set of related information needs, resulting in one or more goals."

example, “exercise” can be supported by queries such as “running shoes,” “best bike routes,” and “personal trainer.”

CrowdPlan

The CrowdPlan algorithm takes a high-level user mission m and generates a simple plan $\mathcal{P}_m$ for accomplishing the mission. A simple plan consists of a set of tuples $(g_i, \mathcal{R}_i)$, where g_i is a goal relevant to the mission and $\mathcal{R}_i$ is a set of resources, e.g., search results, associated with the goal g_i. Figure 4.2 depicts the CrowdPlan algorithm, showing the human-driven and machine-driven operations in grey and white boxes respectively. These operations include:

- **decompose**: given a high-level mission m and a set of previous goals $\{g_1, \dots g_k\}$, this operation generates an additional goal g_{k+1} that is relevant for the mission, but different from already stated goals.

- **rewrite**: given a high-level mission m and a goal g_i, this operation generates a search query q_i for finding web resources that help to achieve the goal g_i.

- **assess**: given a high-level mission m and a set of tuples (g_i, q_i), $i = 1 \dots n$, this operation returns an assessment vector $\vec{a} = \{0, 1\}^n$ where bit i indicates whether the search query q_i is likely to return good search results towards accomplishing goal g_i.

- **filter**: given assessment vectors $\vec{a}_1, \dots, \vec{a}_L$ provided by L workers, this operation aggregates the votes and returns a set of the highest quality search queries to retain.

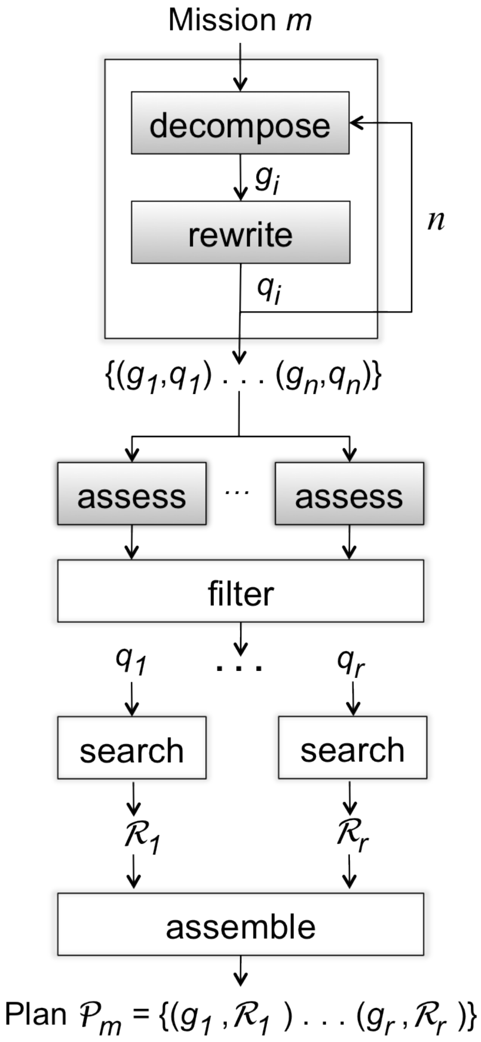


Figure 4.2: CrowdPlan algorithm

- **search**: given a retained search query q_j, this operation retrieves a set of search results $\mathcal{R}_j$ associated with the query.
- **assemble**: this operation returns a simple plan that consists of a set of tuples $(g_j, \mathcal{R}_j)$ to present to the user. Note that this plan can be presented to the user using different forms of visualization.

Each of the human-driven operations (shown in grey in Figure 4.2) – *decompose*, *rewrite*, and *assess* – is associated with a small task that is distributed to Turkers.[2] The *decompose* and *rewrite* operations are combined into a single HIT. A worker is given a high-level mission and a set of existing goals, and is paid $0.10 to first generate an additional goal relevant to the mission and then rewrite the goal as a search query. Combining these two consecutive operations into the same HIT simplifies the problem by allowing a Turker to work off his or her own goal when formulating a query (instead of having to interpret and rewrite someone else's).[3] For each mission, we obtain up to 10 goal-query pairs. The *assess* operation is associated with a HIT that pays a worker $0.10 to cross out any search queries that are unlikely to take a step towards accomplishing the mission and discuss how useful the remaining queries are. Each search query is clickable and links directly to a webpage containing the search results returned by Google for that query.

The machine-driven operations include *filter*, *search* and *assemble*. The *filter* operation eliminates potentially problematic search queries as follows. Each query is assigned a removal score $s_q = n_q + vn_q - vp_q$, where n_q is the number of people who gave a negative assessment for that query, vn_q is the number of people who reviewed the search query (by clicking on the link to bring up the search results) before giving a

[2] We envision that the CrowdPlan algorithm can eventually be embedded as part of collaborative planning websites that have access to tens of thousands of human *volunteers*; but for now, we use Mechanical Turk as a platform to recruit human subjects for our experiments.

[3] Note that in the PlateMate algorithm (Section 2.3.1), we purposely split up the Identify step into two tasks, one for describing food items and another for matching items to a nutrition database. Since these two tasks are conceptually different and can be performed by different workers, keeping them separate simplifies the problem solving. In contrast, combining the decompose and rewrite operations in CrowdPlan allows an individual expressing a goal to continue on that thought to suggest a query, which is natural and simpler than having people interpret goals that others propose.

negative assessment for that query, and vp_q is the number of people who reviewed the search query before giving a positive assessment for the query. By giving more weight to workers who have actually reviewed the search query carefully before providing an assessment, this scoring scheme incorporates not only workers' explicit assessments but also implicit measures of their confidence. We request five *assess* HITs per mission and filter out a query if its score is ≥ 3, which represents a confidence-weighted majority decision. The remaining queries are ranked by their scores in ascending order.

The *search* operation uses the Google Search API to retrieve eight search results for each query. The *assemble* operation then puts together a simple plan, consisting of goals and search results, to display to the user. This step can either collect search results into a list to be displayed, as they would be in a standard search engine, or provide a visualizer for navigating the different results for each goal (e.g., see Figure 4.4(b) on page 106).

The design choices we made in creating this particular algorithm were influenced heavily by our observations about how workers responded to the task. For example, the *decompose* operation could have followed a top-down approach. Workers would first provide a coarse representation of the mission (e.g., "I want to throw a Thanksgiving dinner party") by naming a few goals that encompass the entire solution (e.g., "plan activities," "invite people," and "cook dinner"), then provide successively finer-grained subgoals to accomplish each of the goals. However, in our pilot study, we found that Turkers did not operate at that level of abstraction and often provided goals that did not require further decomposition. Therefore, we made the *decompose*

	New Year's Resolutions / Life Goals
1	cook at home more often
2	manage my inbox better
3	become healthier by working out more
4	run a marathon
5	find an academic job in a good research university in the US
6	become a competitive amateur triathlete.
7	be a good (new) mother
8	start song writing
9	get into petroleum engineering/natural gas field
10	get outside more
11	take a trip to the space
12	be happier
13	lose 80 pounds
14	keep in better touch with high school friends
	Concrete Tasks
1	choose a wedding DJ
2	book a great honeymoon for August 7-14
3	figure out where to go on a week-long sailing vacation with nine friends
4	buy a new pair of dress pants
5	survive the Jan-Feb crazy conference deadlines
6	start exercising and follow an appropriate training program (to become a competitive amateur triathlete)
7	finish the bathroom and laundry room in our basement
8	see if Honda will fix my seatbelt for free
9	kick my friend in the arse
10	find a place to live in Toronto
11	finish Need For Speed Hot Pursuit game
12	shower daily
13	go to the market and buy groceries
14	change address on my car insurance policy

Figure 4.3: Mission statements submitted by subjects

operation more akin to an iterative, brainstorming task in which workers are asked to come up with concrete goals towards accomplishing the mission.

The algorithm is implemented in Javascript and uses TurKit [60] to interface with Mechanical Turk.

Evaluation

In order to evaluate how well our system can answer high-level queries, we recruited a convenient sample of 14 subjects to each give us two mission statements. One mission statement should be in the form of a new year resolution or life goal, and the other should be a concrete task that they want to accomplish. Subjects were mostly recent college graduates who did not major in computer science, and were told that we were working on an information retrieval system that can help answer high-level search queries. They were told that their missions may be shown publicly, but did not know that human computation was involved. Figure 4.3 shows the high-level missions we received, which range from very concrete, actionable tasks (e.g., "change address on my car insurance policy") to less specific, long-term aspirations (e.g., "be happier").

One of the benefits of the simple plans generated by CrowdPlan is that they provide an explanation (in the form of goals) for the search results returned to the user. To study the effect of explanations, for each mission, we asked 10 Turkers to rate the relevance of the CrowdPlan search results on a 4-point scale (0="not helpful", 3="helpful"). Half of the Turkers were given explanations and the other half were not. Workers were paid $0.20 per HIT.

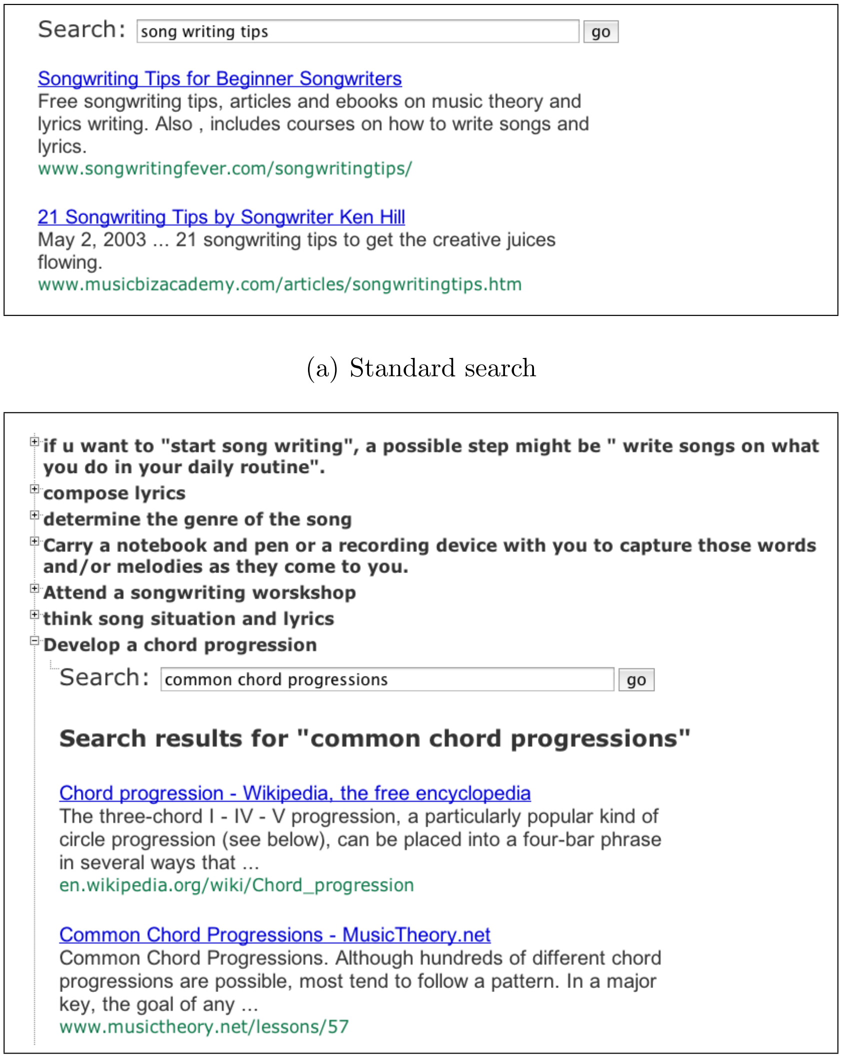
Results show that when given explanations, workers judged the search results to be more relevant. We observe a significant difference ($t(27) = 2.96$, $p < 0.01$) in the average relevance score between the given explanations ($\mu = 1.93$, $\sigma = 0.43$) and the no explanations ($\mu = 1.75$, $\sigma = 0.41$) conditions. We also observe a significant difference ($t(27) = 3.03$, $p < 0.01$) in the discounted cumulative gain [41] between the given explanations ($\mu = 9.7$, $\sigma = 2.25$) and the no explanations ($\mu = 8.72$, $\sigma = 2.24$) conditions.

In looking through the search results, we find that without explanations, the steps for solving a particular problem sometimes appear tangential or even irrelevant. For example, one of the suggested queries for spending time outdoors is “ALTA,” which refers to a non-profit tennis organization. The goal that is associated with this query is actually “take up tennis.” Without this explanation, it is difficult for the user to know why the search result for ALTA would be relevant to his or her high-level mission.

In light of this observation, we created a list-view visualization of the simple plan (see Figure 4.4(b)), which displays the decomposed goals for the mission, the search query associated with each goal, and a short list of five search results. To evaluate the effectiveness of this interface, we asked our 14 subjects to spend three minutes using a standard search engine (Figure 4.4(a)) and then a simple plan in list view (Figure 4.4(b)) to find web resources to help them achieve their missions. This ordering allowed users to search on their own first, without having seen (and be biased by), the goals in simple plans. We then asked subjects to compare the two interfaces in terms of how well each interface helped them accomplish their missions.[4]

We found a split in opinion: seven subjects preferred the simple plan interface over the standard interface, and the other seven preferred the standard interface over the simple plan interface. Subjects who preferred the standard interface commented that it was more “straightforward” to use and generated more “one-stop” search results

[4] A reader interested in additional user studies on CrowdPlan can refer to Law and Zhang [53].



Search: song writing tips go

Songwriting Tips for Beginner Songwriters
Free songwriting tips, articles and ebooks on music theory and lyrics writing. Also , includes courses on how to write songs and lyrics.
www.songwritingfever.com/songwritingtips/

21 Songwriting Tips by Songwriter Ken Hill
May 2, 2003 ... 21 songwriting tips to get the creative juices flowing.
www.musicbizacademy.com/articles/songwritingtips.htm

(a) Standard search

⊞ **if u want to "start song writing", a possible step might be " write songs on what you do in your daily routine".**
⊞ **compose lyrics**
⊞ **determine the genre of the song**
⊞ **Carry a notebook and pen or a recording device with you to capture those words and/or melodies as they come to you.**
⊞ **Attend a songwriting worskshop**
⊞ **think song situation and lyrics**
⊟ **Develop a chord progression**

Search: common chord progressions go

Search results for "common chord progressions"

Chord progression - Wikipedia, the free encyclopedia
The three-chord I - IV - V progression, a particularly popular kind of circle progression (see below), can be placed into a four-bar phrase in several ways that ...
en.wikipedia.org/wiki/Chord_progression

Common Chord Progressions - MusicTheory.net
Common Chord Progressions. Although hundreds of different chord progressions are possible, most tend to follow a pattern. In a major key, the goal of any ...
www.musictheory.net/lessons/57

(b) Simple plan in list view for the mission "start song writing"

Figure 4.4: Standard Search versus Simple Plan

(i.e., general purpose websites with links to resources), while simple plans generated some search results that were irrelevant to what they were looking for specifically. Here are some comments:

– I like the idea behind simple plans, but I find it more straightforward to use a regular search tool.

- “The standard search tool was better because I knew enough about what I wanted that I could type in more specific searches.”
- “I think many good websites will give me a one-stop shop for marathon information. The simple plan was fairly comprehensive although perhaps too specific.”

In contrast, subjects who preferred simple plans over standard search results had the following comments:

- “The simple plan actually organized my search for me, into discrete and doable steps. The standard search tool left me to do all the creative parsing and generation of search terms. I felt that the simple plan gave me a roadmap to the entire space by my mentioning something in that space.”
- “The simple plan gave me some good ideas for concrete steps to take that would help me accomplish my goal. Therefore, the search queries were more focused, and the overall process more effective.”
- “The simple plan gave me a birds-eye view of useful search queries from which to pick. the recommendations were really useful. My reaction to some of them was ‘oh, I didn’t think of that. good point!’ The simple plan solves to some degree the problem of unknown unknown, which is that in order to find something you need to know you need it. This problem makes the standard interface of limited use, because you need to know a priori what you have to do in order to find instructions on how to do it. But the simple plan, being broader in its results, suggests things you didn’t think of.”

These comments are revealing for several reasons. First, they suggest that not all missions require decomposition. For some missions, a standard search engine may already be quite good at retrieving relevant results for well-specified search queries that rephrase a mission statement. Second, they suggest that simple plans can be effective in three ways—making users aware of aspects of the mission they had not originally thought of, providing an organized roadmap of relevant goals, and suggesting concrete, actionable steps towards accomplishing the mission.

4.4 Discussion

The 8-puzzle experiment and the CrowdPlan system show that having crowds guide the problem-solving process and synthesize plans can lead to effective solutions and novel applications. In constructing interfaces, workflows, and communication mechanisms that involve the crowd in more general problem solving, we remain sensitive to the concern that individuals in the crowd may only make small contributions and that some contributions may be noisy. Understanding how to design effective patterns of interactions for control and synthesis is an important area for future research, and should draw on our understanding of the crowd's ability to perform control and synthesis related actions such as suggesting subgoals and collating ideas.

We find that in both worker-worker and worker-requester interactions, being able to effectively share and present problem-solving context is crucial. In the 8-puzzle, we observed that short messages about problem-solving strategies were easy to process and of high value when good paths were identified. We saw examples of effective reuse when messages were edited and passed on, and also examples in which workers identified new strategies when they found suggestions unhelpful. For workers to make such evaluative decisions and act effectively, the problem-solving context provided by workers and the system needs to be easily understandable.

In CrowdPlan, we found that search results were judged to be significantly more relevant when presented alongside the goals for which they were generated. As CrowdPlan tends to return search results covering a diverse set of issues related to the mission, a potential drawback of the increased diversity is decreased comprehensibility. This suggests that adding additional steps to the CrowdPlan algorithm aimed at improving clarity may improve the usability of the system. From our subjects, we also learned that CrowdPlan sometimes missed out on useful context that was known to the requester but not shared with the workers. As an example, for the mission "I want to get outside more," CrowdPlan returned search results for taking up gardening, birdwatching, taking daily walks, geocaching, and adopting a dog. But when presented with these results, the subject commented that he was looking for "websites geared toward more active outdoor activities in natural surroundings." This suggests that sharing additional context (e.g., allowing for richer missions as in Mobi), or allowing for more back-and-forth between the requester and the workers, may enable CrowdPlan and other collaborative planning systems to better tailor solutions to each user.

As we move toward crowdsourcing general computation, the notion of expertise becomes more prominent as the roles people play become more diverse and specialized. The ability to identify expertise and reward individuals for providing meta-expertise (e.g., controlling the algorithmic process, routing to others who are experts), may allow us to solve problems that we otherwise would not be able to solve with a crowd. The next chapter introduces methods for *task routing*, that aim to harness the ability of people to both contribute to a solution and guide the problem-solving process by routing tasks to others who they believe can effectively solve and route.

Chapter 5

Task Routing

Engaging a crowd to tackle complex tasks relies not only on effective coordination, but on recruiting individuals with relevant expertise to join the problem-solving effort. One approach for bringing expertise to tasks is to pool knowledge about people's competencies and preferences and assign tasks in a centralized manner. Another approach is to rely on individuals in a system to select tasks themselves. Both approaches have flaws. In the former, a system may not know which individuals have the required expertise. In the latter, while individuals are often able to gauge their own expertise, they may not know which tasks best match their respective competencies.

In social networks, an individual's knowledge extends beyond their own expertise on tasks and topics to knowledge about the expertise of others. Members of a social network may know who among their friends can best answer a particular question or provide valuable opinions on a topic of discussion. Even in situations where an individual cannot identify an expert who can best contribute to a task, they may know people who would likely know experts. They may also be able to identify subsets of individuals who share a particular interest, among whom the requisite expertise is likely to exist.

We are interested in principles and methods for *task routing* that draw on the distributed intelligence of individuals across a social network. The idea is to harness the ability of people to contribute to a solution *and* route tasks to others who they believe can effectively solve and route. Task routing provides a paradigm for problem solving in which individuals in a crowd become engaged with tasks based on their peers' assessments of their expertise. On the task level, effective task routing aims to take advantage of people's knowledge about solving problems as well as people's knowledge about others' abilities to contribute. People make routing decisions in a peer-to-peer manner, and the system rewards participants for their contributions, both direct and indirect through routing. On the organizational level, task routing may provide a means for bringing tasks to individuals effectively, where people's routing decisions take into account not only an individual's expertise on the particular task, but also their ability to contribute as a router.

In this chapter, we focus on the special case in which the task is to obtain an accurate probability assessment about an uncertain event. The task is passed among individuals in a network, and each participant can update the posterior probability and forward the task to a neighbor. We introduce *routing scoring rules* for incentivizing contributions. Given an assumption of common knowledge about the network structure and the amount of information held by everyone in the network, truthful reporting of posterior probability assessments and optimal routing can be obtained in a Perfect Bayesian Equilibrium. While this result is theoretically sound, optimal routing is NP-hard, which suggests that people may have difficulty computing routing decisions in practice. The common knowledge assumption is also unlikely to hold for large social networks, where each person's information about the competencies of others is limited to a local neighborhood (e.g., friends, and perhaps friends of friends).

To address these concerns, we consider designing incentive schemes for task routing problems where knowledge about the network structure and others' abilities is limited to an individual's local neighborhood. The main contribution is the introduction of a family of *local routing rules*, that isolate simple routing decisions in equilibrium under local knowledge about others' expertise and take advantage of such local knowledge to promote effective routing decisions. We achieve this by incentivizing participants to make routing decisions based on short, locally optimal paths that can be computed easily using local knowledge. In essence, *we design incentive schemes that explicitly enable equilibrium behavior for which the inference required of participants is tractable.*[1]

We provide a full characterization of local routing rules, and show that they are the only routing scoring rules that induce truthful equilibria in which best responses are invariant to knowledge outside of a local neighborhood. Simulation results demonstrate that equilibrium routing strategies based on local routing rules lead to effective information aggregation.

[1] This is analogous to the role of strategy-proofness in simplifying strategic problems facing agents in mechanism design [71].

5.1 Related Work

Methods for automated and manual routing of tasks have been employed in real world online networks. For example, question-answering services such as Aardvark [31] allow a user to ask questions in natural language, which the system interprets and automatically routes to appropriate individuals in the user's social graph based on an assessment of who is best able and willing to provide an answer. Aardvark also allows for peer routing; a user can manually route questions to others, which enables the system to reach users outside its fund of knowledge about people and their expertise. Aardvark does not explicitly reward contributions, and instead relies on people's goodwill and social connections. In studying incentive mechanisms for task routing, we are exploring how properly rewarding participants for their contributions can help promote contributions to problem solving and routing more broadly.

Leveraging individuals' abilities to both solve and spread the word about the task was a key component of the winning team's strategy in the DARPA Red Balloon Challenge [73]. The task was to find large helium-filled balloons placed in ten undisclosed locations across the continental United States. The winning team introduced an incentive mechanism that uses a limited budget to incentivize individuals to look for balloons and to let their friends know about the task.[2] This mechanism aims to induce participants to broadcast the task to everyone they know, and is well-suited for one-off, high-stake scenarios such as search and rescue operations for which the benefit of reaching a large audience significantly outweighs the cost of people's atten-

[2] The interested reader may refer to Emek et al. [26], Douceur and Moscibroda [22], and Drucker and Fleischer [24] for related theoretical analysis, and related work on query incentive networks [48, 4, 19] that analyze games in which players split rewards to recruit others to answer a query.

tion. In contrast, the mechanisms in our work aim to leverage the expertise within a network by bringing to people's attention the tasks that they can best contribute to. These mechanisms are well-suited for efficiently processing a stream of tasks, without overloading people with information on every task.

The problem of task routing is also related to the problem of decentralized search on networks, in which the goal is to find a target node quickly through local routing decisions [92, 21, 99, 47, 1]. In such work, the goal is to identify a *single* target node representing a particular individual. While this differs from our task routing problem, the results still provide theoretical and experimental support for the prospect that routing decisions with local information may have effective global performance.

One can view routing scoring rules as an extension of market scoring rules [29], which provide proper incentives for individuals participating in a prediction market to improve probability estimates by contributing additional information. The major difference between task routing and a prediction market is in who takes on the burden of identifying expertise. While a prediction market places the responsibility on individuals to find prediction tasks for which they have useful information, task routing incentivizes individuals to notify others with appropriate expertise who may otherwise be unaware of the task.

5.2 Task Routing for Prediction Tasks

To formalize the setting, consider a single prediction task T, for which we would like to gather an accurate probability assessment of the true state $\omega \in \Omega$. The probability assessment task can be for any state of the world that will be revealed later in time, e.g., "Will it snow next Tuesday in Boston?" or "Will the Boston Celtics win the NBA championship this year?" We consider discrete state spaces, and assume without loss of generality a binary state space, such that $\Omega = \{Y, N\}$.[3]

Consider a *routing game* with n players, where each player is represented by a node on the *routing graph* $G = (V, E)$. Edges in the graph may be directed or undirected, and indicate whether a particular player can route the task to another player. The task is initially assigned to a source player named player 1, with later players on a routing path numbered sequentially. The source player may be determined by the system or by the individual posting the task. The source player is asked to update the probability of state Y from the prior probability p^0 to some probability p^1, and in addition, to route the task to a neighbor. The selected neighbor is then asked to update the assessment p^1 to p^2 and route the task to a neighbor, and so on, until the game ends after a prespecified number of rounds R, when a final assessment must be made. We assume players receiving the task are provided with a list of people who have participated so far, as well as the number of rounds that remain. Players are allowed to route to players who have participated thus far, but know that past participants may not have any additional information to contribute and may only be able to help with routing. *Our goal is to arrive at an accurate assessment after R rounds by designing incentive mechanisms that will induce each player to update probability assessments truthfully and route the task to other players that can best refine the prediction.*

[3] For an event with more than two states, the task is to gather a probability vector with a likelihood assigned to each state. We can handle such events by using multi-class versions of proper scoring rules, and all of our results extend straightforwardly.

We model players' knowledge about the task as follows: the true state of the world is drawn according to the probability distribution $\Pr(Y) = p^0$ and $\Pr(N) = 1 - p^0$, which is common knowledge to all players. While no player observes the true state directly, each player may receive additional information about the true state. To model this state of affairs, each player privately observes the outcome of some number of coin flips drawn according to a commonly known distribution that depends on the true state. Different players may observe different numbers of coin flips, where players observing more coin flips are *a priori* more knowledgeable.

Formally, we represent player i's signal c_i as a random bit vector of length l_i, where bit c_{ik} is a random variable over the outcome of the k-th coin flip observed by player i. We assume the value of bits of signal are *conditionally independent* given the true state, and drawn from the same distribution (known to all players) for all players and all bits, such that $\Pr(c_{ik} = H|\omega) = \Pr(c_{jm} = H|\omega)$ for all players i, j, bits k, m, and realization H (head). Each bit of signal is assumed to be *informative*, that is, $\Pr(c_{ik} = H|\omega = Y) \neq \Pr(c_{ik} = H|\omega = N)$ for all i, k. We also assume that bits of signal are *distinct*, that is, $\Pr(\omega = o|c_{ik} = H) \neq \Pr(\omega = o|c_{ik} = T)$ for all i, k, o, where H is heads and T is tails.[4] We assume the realization of each player's signal is private, and make different assumptions about the knowledge of a player about the *number* of coin flips of another player.

With conditionally independent signals, each player can properly update the posterior probability without having to know the signals of previous players or their

[4] These assumptions rule out degenerate cases and can be made without loss of generality. A signal that is not informative can be removed from the signal space, and two signals that are not distinct can be treated as the same signal.

length, as long as previous updates were done truthfully [14]. The posterior incorporates, and sufficiently summarizes, all information collected thus far. To perform updates, players need to only know the signal distribution with respect to their own signal, which we assume is known to all players. This is useful practically in that players do not have to keep track of nor communicate their signals, and can simply report an updated posterior probability.

5.3 Routing Scoring Rules

With rational, self-interested players who have no intrinsic value (or cost) for solving or routing a particular task, ensuring effective task routing requires mechanisms that will incentivize players to both truthfully update posterior probabilities and route tasks to individuals who can best refine the predictions of the tasks. In this section, we review strictly proper scoring rules and market scoring rules for incentivizing truthful reports, and introduce routing scoring rules, which also incentivize effective routing decisions.

In the forecasting literature, *strictly proper scoring rules* [83] are mechanisms that strictly incentivize a forecaster to truthfully reveal his subjective probability of an event, typically under the assumption that participants are risk neutral. The outcome of the event is assumed to be observable in the future, and payments are conditioned on the outcome. A well-known strictly proper scoring rule is the *quadratic scoring rule*, under which a player reporting probability q for state Y is rewarded $1-(1-q)^2$ when the true state is Y and $1-q^2$ when the true state is N. Other well-known strictly proper scoring rules include the logarithmic and spherical scoring rules. Any strictly proper scoring rule can be scaled or normalized via linear transformations to form another strictly proper scoring rule [7].

Market scoring rules [29] extend strictly proper scoring rules to settings where we wish to aggregate information across multiple people. Given a sequence of reports, player i reporting p^i is rewarded $s_i - s_{i-1}$, where s_i denotes the score of player i as computed by some strictly proper scoring rule applied to this player's report. Note that since strictly proper scoring rules incentivize accurate reports, a player's reward under a market scoring rule is positive if and only if he improves the prediction.

Building on market scoring rules, we introduce *routing scoring rules* to incentivize accurate predictions, along with effective routing decisions.

Definition 5.1. *A* **routing scoring rule** *defines a sequence of positive integers k_1, …, k_{R-1}, which rewards players $i \in \{1, \ldots, R-1\}$ on the routing path:*

$$(1-\alpha)s_i + \alpha s_{i+k_i} - s_{i-1} \tag{5.1}$$

where s_i is the score under an arbitrary strictly proper scoring rule, $\alpha \in (0,1)$ is a constant, and $i + k_i \leq R$ for all players i. Player R reports but does not route and is paid $s_R - s_{R-1}$.

In a routing scoring rule, player i's payment is based on the marginal value the player provides for refining the prediction, as measured by a combination of his report and the report of the player who receives the task k_i steps after him, relative to the report of the player just before him. For player 1, s_0 denotes the score computed with respect to the prior p^0. Each player i can be paid for up to $R-i$ steps forward, and the final player R does not route and is paid by the market scoring rule $s_R - s_{R-1}$. Players who participate multiple times within a routing game are paid based on the routing scoring rule the first time they receive the task, and paid by the market scoring rule in any subsequent interactions.[5]

Intuitively, routing scoring rules reward players who are experts as well as players who are knowledgeable about the expertise of other players. We introduce several routing scoring rules of particular interest. We first consider the *myopic routing scoring rule* (MRSR), which sets $k_i = 1$ for all players $i < R$. This routing scoring rule aims to reward a player for submitting accurate probability assessments and routing in a greedy manner to the adjacent player who can most accurately refine the probability assessment.

Lemma 5.1. *Consider a routing game in which each player participates at most once. The total payment from the system with MRSR is* $s_R - s_0 + \alpha(s_R - s_1)$.

The lemma follows from taking telescoping sums, and states that, for MRSR, the center needs to only pay for the difference between the final assessment and the initial assessment, since each player is only paid for the additional information they provide and their routing decision. The expression is familiar from market scoring rules, containing just an additional term due to routing payments.

We can extend the MRSR to reward players' routing decisions based on the accuracy of information after $k_i = \min(k, R - i)$ more players have provided their information. The *k-step routing scoring rule* (kRSR) rewards a player based on his report, as well as the eventual consequence of his routing decision k steps into the

[5] For the local knowledge settings we consider later in the chapter, this avoids situations in which a player may try to hold on to a task by making suboptimal routing decisions that lead to their being routed the task again, with the intent of earning multiple routing payments beyond the first.

future. Unlike MRSR, kRSR rewards players for routing to players who may not have information themselves, but who are still able to route to others who do.

In particular, when player i's routing payment is based on player R's score, that is, $i + k_i = R$, for all i, we call this the *path-rewarding routing scoring rule* (PRSR). As its name suggests, this routing scoring rule seeks to focus a player's attention on the final consequence of his routing decision, judged at the end of the game.

The choice of routing scoring rule affects players' routing decisions in equilibrium, which in turn affect how much information is aggregated. To formally establish the connection between a player's score and the amount of information aggregated, we show that the expected score is strictly increasing in the total number of coin flips collected:

Lemma 5.2. *Let S' and S'' denote two possible sequences of players through the first k rounds of the routing process that are identical up to player $i < k$. Assume all players truthfully update posterior probabilities, and that player i knows the number of bits l_j for players $i < j \leq k$ on S' and S''. Let $E^i_S[s_k]$ denote player i's expectation, taken immediately after his own report, of the score after player k's report in path S. $E^i_{S'}[s_k] > E^i_{S''}[s_k]$ holds if and only if $\sum_{m \in u(S')} l_m > \sum_{n \in u(S'')} l_n$, where $u(S)$ is the (unique) set of players in S.*

Proof. (sketch) Assume without loss of generality that there are a total of n coin flips in S', and $n + m$ coin flips in S'', $m > 0$. The expected score of player k from S'' consists of two (hypothetical) components: (a) the score he would get when giving a prediction after receiving the first n coin flips, denoted $s_{[n]}$, and (b) the difference in the score he would get by changing his prediction after receiving the next m coin flips, denoted $s_{[n+m]} - s_{[n]}$. The expectation of the first part is the same as the expected score of player k from S', and the expectation of the second component is always non-negative given any strictly proper scoring rule. □

Intuitively speaking, additional bits of information can only improve the accuracy of the prediction in expectation. Since strictly proper scoring rules reward accuracy, collecting more coin flips will lead to higher scores in expectation.

5.4 Common Knowledge

Having introduced routing scoring rules of interest, we consider an equilibrium analysis of the associated routing game. We first consider the case where the network structure and the number of coin flips l_i observed by each player i is common knowledge. Note the actual signal realizations are still assumed private.

5.4.1 Clique Topology

Let us first consider the routing game on a *clique*, where each player can route the task to any other player. Given the clique topology, an optimal routing algorithm can just route myopically and collect as many coin flips as possible at each step. In a clique, there is no opportunity cost for being greedy in this way. We have the following equilibrium result:

Theorem 5.1. *Assume the number of coin flips of each player is common knowledge and that players are risk neutral. Consider a routing game in which the routing graph is a clique, and let $S_{>i}$ denote the set of players who have yet to receive the task after*

i rounds. Under the myopic routing scoring rule, it is a Perfect Bayesian Equilibrium (PBE) for each player i to truthfully update the posterior probability, and to route the task to player $i+1 \in \operatorname{argmax}_{m \in S_{>i}} l_m$, with the belief that all other players update the posterior probability truthfully.

Proof. (sketch) We show that no player wishes to deviate from the equilibrium strategy, given the belief that all other players report truthfully. For any player i, we first show that player i should honestly update the posterior beliefs by establishing that (a) truthful reporting maximizes s_i, and that (b) for any player m who may be routed the task, truthful reporting by player i maximizes the score s_m. Note that for (a), since s_i is based on a strictly proper scoring rule, truthful reporting maximizes the expectation of s_i. For (b), the expected score of s_m (from the perspective of player i) is strictly greater when player i reports honestly because s_m is based on a strictly proper scoring rule. It is left to show that player i maximizes s_{i+1} by routing to the player in $S_{>i}$ with the most coin flips; this follows from Lemma 5.2. □

5.4.2 General Networks

We now consider routing games on general networks with missing edges; e.g., only managers can route tasks between teams and only friends can route to friends. We can state the algorithmic problem of finding the optimal route in terms of collecting coin flips:

Problem 5.1. *Consider the routing graph $G = (V, E)$, in which nodes are assigned non-negative integer weights w_i (coin flips). Given a starting node o, find a path of length at most k such that the sum of weights on the path is maximized.*

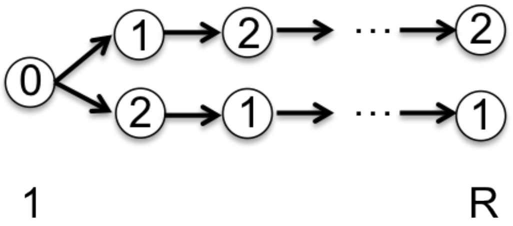


Figure 5.1: A routing game for which myopic routing (along the bottom path) is suboptimal. Numbers in nodes are the number of coin flips held by each player.

Note that a player can route to another player who has received the task before (the path need not be *simple*), but no additional information is collected in subsequent visits to the same player.

Immediately, we see that myopic routing will not always find the optimal solution to Problem 5.1, as routing to the neighbor with the most coin flips does not consider the effect this can have on future routing decisions, and can now convey an opportunity cost. Figure 5.1 illustrates an example in which myopic routing would lead player 1 and all subsequent players to route along the bottom path, while the optimal solution requires players to route along the top path.

We can show that this problem is NP-hard for variable path length k:

Lemma 5.3. *Problem 5.1 is NP-hard.*

Proof. Consider a reduction from the Hamiltonian Path problem. Let all nodes have weight 1, and set $k = |V|$. The solution path has total weight $|V|$ if and only if all nodes are visited within k steps, that is, a Hamiltonian Path exists. □

While the problem is NP-hard for a variable path length k, for small constant k the optimal path may be tractable to compute via exhaustive search.

But intractability is not the only difficulty we face. Even if players can compute the optimal path, we still need to find incentives that induce players to honestly report their information and to route along the optimal path. The path-rewarding routing scoring rule does just that.

Theorem 5.2. *Assume the number of coin flips of each player is common knowledge and that players are risk neutral. Let $S_{>i}$ denote the set of players who have yet to receive the task after i rounds. Let Q_i denote a solution to problem 5.1 for which $k = R - i$, $o = i$, and $w_m = l_m$ if $m \in S_{>i}$ and $w_m = 0$ otherwise. Under the path-rewarding routing scoring rule, it is a PBE for each player i to truthfully update the posterior probability and route the task to the next player in the path provided by Q_i, with the belief that all other players follow this strategy.*[6]

Proof. (sketch) Using similar arguments as in the proof sketch for Theorem 5.1, we show that no player wishes to deviate from the equilibrium strategy, given the belief that all other players report truthfully. For any player i, we first show that player i should honestly update the posterior beliefs by establishing that (a) truthful reporting maximizes s_i, and (b) for any subsequent sequence of players $i+1, \dots, R$ who may be routed the task, truthful reporting by player i maximizes the score s_R at the end. For (a), since s_i is based on a strictly proper scoring rule, truthful reporting maximizes the expectation of s_i. For (b), the expected score of s_R (from the perspective of player i) is strictly greater when player i reports honestly because s_R is based on a strictly proper scoring rule.

[6] In this setting, a player who participates multiple times does not receive, nor require, any incentives for routing beyond the first time. This is because routing along an optimal path is required for maximizing the expected score at the end of the game, which is the basis for a player's (first time) routing payment under the path-rewarding routing scoring rule.

It is left to show that player i maximizes s_R by routing to the next player in the path provided by Q_i; this follows from Lemma 5.2. □

Since PRSR rewards each participant's routing decision based on the final score, it is in each participant's interest to maximize the number of coin flips collected along the entire routing path. We can show that reporting honestly and routing this way is the only behavior that can be supported in equilibrium under PRSR:

Theorem 5.3. *The set of PBE identified in Theorem 5.2 (corresponding to possible ties in the solution to problem 5.1) are the only PBE of the routing game under PRSR.*

Proof. (sketch) Given any routing path, by backward induction every player should update the posterior probability truthfully because participants' scores are computed using a strictly proper scoring rule. Given that players update truthfully, by backwards induction every player i should route along the path identified by some solution Q_i because maximizing the number of coin flips collected maximizes the routing portion of each player's score (Lemma 5.2). □

5.5 Local Common Knowledge

Although people may know one another's expertise in small organizations, the common knowledge assumption becomes unreasonable for larger organizations and social networks. An individual will not necessarily know everyone else, and may only have limited information about the expertise and connectivity of individuals outside of a local neighborhood.

We replace the common knowledge assumption with a requirement that individuals all attain the same minimal level of knowledge about each other's expertise within a local neighborhood of a particular size, defined by the number of hops between participants. For example, all friends of a particular person are aware of his expertise (one hop). Friends of his friends may also be aware (two hops).

Definition 5.2. *A routing game satisfies the* **local common knowledge assumption within m-hops** *if, for all nodes (individuals) i, (a) l_i is common knowledge to all individuals connected to i via some path of length at most m, and (b) i knows all paths of length at most m connecting i to other individuals, and this is common knowledge.*

For example, 1-hop local common knowledge assumes that all friends of a particular person know the person's level of expertise, and 2-hop local common knowledge extends this shared knowledge to his friends of friends. Note that the local common knowledge assumption within m-hops is just a minimal requirement and does not preclude a player from having more information.

Given that a player may only have m-hop local common knowledge, let's consider the problem facing such a player when deciding how to route to maximize the final prediction quality after R steps. Routing optimally may require the player to use the history of routing decisions to infer why certain people were not routed the task (but could have been), based on which to perform inference about the amount of information held by different people in the network. Furthermore, optimal routing requires a player to make inferences about the values that can be generated from the routing decisions of subsequent players beyond his locality. Not only is such reasoning complex and likely impractical, any equilibrium to induce optimal routing will likely be fragile because it requires players to adopt priors over other players' beliefs.

An attempt to avoid such issues may suggest rewarding players based on a m-step routing rule whenever the local common knowledge assumption holds for m-hops. The problem with this suggestion is that a player would still have to consider the routing decisions of players outside his locality because maximizing his payoff requires considering the routing decisions of the chain of players within his locality. For example, consider the two-step routing rule (see bottom of Figure 5.2). For any player, the score two steps forward will depend in part on the routing decision of the next player. But since the next player is paid for the score two steps forward (from him), his routing decision will depend not only on the amount of information held by the player after him, but also that player's routing decision. Since each player has to consider the routing decision of the next player, each player has to reason about the future routing decisions of all players down the routing path, in order to just compute the expected score after two steps.

This motivates the family of *local routing rules*, under which players' strategies in equilibrium rely only on computations based on local information. We define the notion of a local strategy as follows:

Definition 5.3. *A player i in a routing game adopts a* **m-local strategy** *if his routing decision depends only on m-hop local common knowledge and is invariant to any beliefs the player might have about players outside of his own locality.*

Let us first consider the following local routing rule, designed to be useful with 2-hop local common knowledge:

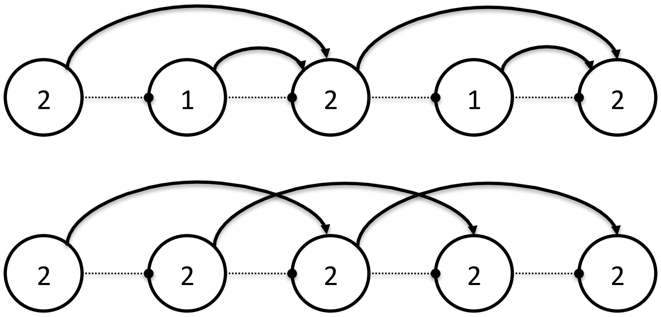


Figure 5.2: Illustration of the 2-1-2-1 and 2-step routing rules. Arrows depict dependencies in routing payments.

Definition 5.4. *The* **2-1-2-1 routing rule** *is a routing scoring rule which sets* $k_i = 2$ *if* i *is odd and* $i < R - 1$*, and* $k_i = 1$ *otherwise.*

The 2-1-2-1 routing rule incentivizes players to compute locally optimal paths of length two (see top of Figure 5.2), which can be computed with local common knowledge. As even-numbered players are paid based on the myopic routing scoring rule, they will route to the available player with the most number of coin flips. Since each odd-numbered player knows the number of coin flips that can be collected from the next even-numbered player and the next odd-numbered player that is routed the task, he can compute the best local path without regard to routing decisions beyond his locality. Players still need to take into account which other players have already participated, but no other inference based on history is necessary.

Expanding on the idea, we construct a class of routing scoring rules (e.g., MRSR, 2-1-2-1, 3-2-1-3-2-1, ...) that incentivize players to compute *locally optimal paths* for m-hop local common knowledge.

Definition 5.5. *The* $\boldsymbol{m}$**-hop routing rule** *is a routing scoring rule which sets* $k_i = \min[m - (i-1) \bmod m, R - i]$.

The m-hop routing rule supports the following equilibrium behavior:

Theorem 5.4. *Assume that players are risk neutral and m-hop local common knowledge holds. Let $S_{>i}$ denote the set of players who have yet to receive the task after i rounds. Let Q_i denote a solution to problem 5.1 for which $k = \min[m - (i - 1) \bmod m, R - i]$, $o = i$, and $w_j = l_j$ if $j \in S_{>i}$ and $w_j = 0$ otherwise. Under the m-hop routing rule, it is a PBE for each player i to truthfully update the posterior probability and route the task to the next player in the path provided by Q_i, with the belief that all other players follow this strategy.*

Proof. (sketch) Using similar arguments as the proof sketch for Theorem 5.1, we can show that players should truthfully update the posterior probability. To show player i should route based on Q_i, we first note that Q_i is computable given m-hop local common knowledge. Since Q_i maximizes the number of coin flips collected in the next k steps, Lemma 5.2 proves the point, and the theorem. □

Unlike in the common knowledge setting under the path-rewarding routing scoring rule, this equilibrium under the m-hop routing rule may not be unique. For a player routing more than once, after the first time, the player is weakly indifferent among all routing decisions because his payment reduces to the market scoring rule for subsequent routing opportunities. Such a player need not route along a locally optimal path in making subsequent routing decisions and this can affect the equilibrium behavior of other players who may route the task back to this player. If we wish to ensure that routing along a locally optimal path is a unique equilibrium, we can modify the routing game slightly to prevent players from routing to other players who have already participated in the game.[7]

The main idea behind the m-hop routing rule is that each player can compute his best routing action with respect to the decisions in his locality and without regard to routing decisions beyond his locality. It turns out that this property can be satisfied by other local routing rules as well. For example, when $m = 3$, the 3-1-1-3-1-1 routing rule is one in which the first of three players in sequence is paid by the score three steps forward, but the next two players are each paid myopically. Note that the first player here can still compute his optimal routing decision using only local common knowledge by computing the routing decisions of others in his locality via backwards induction. We can thus characterize the entire family of local routing rules:

Definition 5.6. *Given m-hop local common knowledge, the family of* ***m*-local routing rules** *consists of routing scoring rules defined with parameters* $k_1, \ldots, k_{R-1}$*, that satisfy* $k_{i+j} + j \leq m$ *for all* i *and* $0 \leq j < k_i$*.*

Generally, we can refer to these as *local routing rules*, dispensing with the m when this detail is unimportant. The condition ensures that local routing rules can only reward players whose routing decisions may affect the payoff of an earlier player based on the routing decisions of future players that are within m hops of that earlier player. In other words, it considers the set of routing scoring rules for which the payment to any player should only depend on the local information that player is guaranteed to hold. For example, the 2-1-2-1 routing rule satisfies this condition for $m = 2$ because

[7] We can modify the routing game so that in cases when a player has no one to route to, no routing payments will be assigned. The task is returned to the system which will randomly select a new participant. Since players cannot participate twice in this modified game, uniqueness of the equilibrium stated for the m-hop routing rule in Theorem 5.4, and more generally for local routing rules in Theorem 5.5, can be recovered without requiring further assumptions. The argument is similar to that in the proof of Theorem 5.3.

for an odd i, $k_i \leq 2 \leq m$ and $k_{i+1} + 1 = 2 \leq m$, and for an even i, $k_i = 1 \leq m$. However, the two-step routing scoring rule violates the condition, because for all $i < R-2$, $k_{i+1}+1 = 3 > m$. Note that the m-hop routing rule satisfies the condition, since k_i is set such that $k_{i+j} + j = m$ for all appropriate i and j in Definition 5.6.

We argue that using a local routing rule is necessary and sufficient for the existence of an equilibrium under m-hop local common knowledge, in which participants follow m-local, truthful strategies. We first show sufficiency:

Theorem 5.5. *Assume that risk neutrality and m-hop local common knowledge holds. For any node i and possible path $n_{i+1}, \ldots, n_{i+k_i}$ from i, let the weights w_j on node j be l_j if j has yet to be visited up until then, and 0 otherwise. For any m-local routing rule, consider the following dynamic program:*

$$\begin{aligned} V(n_{j+1}, \ldots, n_{j+k_j} | n_1, \ldots, n_j) = \max_{j+1,\ldots,j+k_{j+1}} [\sum_{b=1}^{k_{j+1}} w_{j+b} \\ + V(n_{j+k_{j+1}+1}, \ldots, n_{j+k_j} | n_1, \ldots, n_{j+k_{j+1}})] \\ V(\emptyset | n_1, \ldots, n_{j+k_j}) = 0 \quad \forall n_1, \ldots, n_{j+k_j} \end{aligned} \tag{5.2}$$

Let $n^*_{i+1}, \ldots, n^*_{i+k_i} = \operatorname{argmax} V(n_{i+1}, \ldots, n_{i+k_i} | n_1, \ldots, n_i)$ denote a solution of the dynamic program. It is a PBE for each player i to truthfully update posterior probabilities and to route the task to n^*_{i+1}, with the belief that all other participants follow this strategy.

Proof. (sketch) To prove the theorem, we first note that all players would truthfully update the posterior probability along the path as we had previously argued, as doing so maximizes the scores computed, based on a player's own assessment and the assessments collected from those routed the task via the routing payment. Second, as

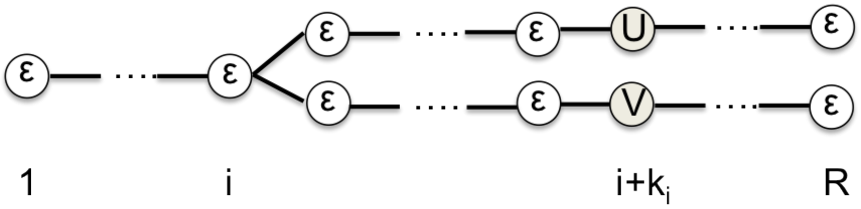


Figure 5.3: Routing game construction for the $j = 0$ case.

the variables and parameters of the dynamic program are only the nodes in paths of length at most k_i from i, and by the definition of local routing rules $k_i \leq m$, players follow m-local strategies. That is, the information that each player i needs to compute the dynamic program is within m hops and thus known to player i. Finally, given the routing decisions of others down the path, the number of coin flips collected is by definition maximized by the routing decisions along the computed path. Applying Lemma 5.2 proves the point, and the theorem. □

Theorem 5.6. *The only routing scoring rules that induce for every routing game a truthful PBE (where players honestly update probability assessments) in m-local strategies are local routing rules.*

Proof. (sketch) Assume for sake of contradiction that there exists a routing scoring rule that induces a truthful PBE for all routing games in m-local strategies but is not a local routing rule. Since this routing scoring rule is not a local routing rule, there must be some i in the sequence for which there exists some j such that $k_{i+j} + j > m$, $0 \leq j < k_i$. Consider the first such i and j.

First consider the case where $j = 0$. We construct a graph with two paths (top and bottom), as shown in Figure 5.3.

Based on the construction, consider two routing games G and G'. In game G the coin flips held by U and V are 1.5ϵ and 1.6ϵ respectively, and in game G' the coin flips at U and V are reversed. Due to the violation of the condition for local routing rules at i for $j = 0$, by construction U and V are more than m hops from player i. In a PBE with m-local strategies, it is thus necessary for the routing decisions of player i to be independent of the number of coin flips held by players at U and V, that is, for the routing decision to be the same for these two games G and G'.

We show that player i's best response to the equilibrium strategies of the other participants depends on G or G'. For both games, using backwards induction, all players strictly prefer to route the task forward (to the right) instead of backwards at any given point in time and for any lookahead depth as induced by their routing payment. This is because a player's expected payment is based on the number of coin flips collected and one can always collect more coin flips in the forward direction (for any player, going backwards would necessitate visiting a node that's been visited before with no new coin flips to share). Since in game G player i would collect more coin flips by routing up due to the higher value at U over V and the reverse is true in game G', player i's best response would be different, which contradicts our assumption.

Now consider the case where $j > 0$. We construct a graph with three paths (top, middle, and bottom), as shown in Figure 5.4.

Based on the construction, consider two routing games G'' and G'''. In game G'' the coin flips held by A, B, and C are ϵ, ϵ, and ϵ respectively, and in game G''' are ϵ, 1.7ϵ, and 1.7ϵ, respectively. Due to the violation of the condition for local routing

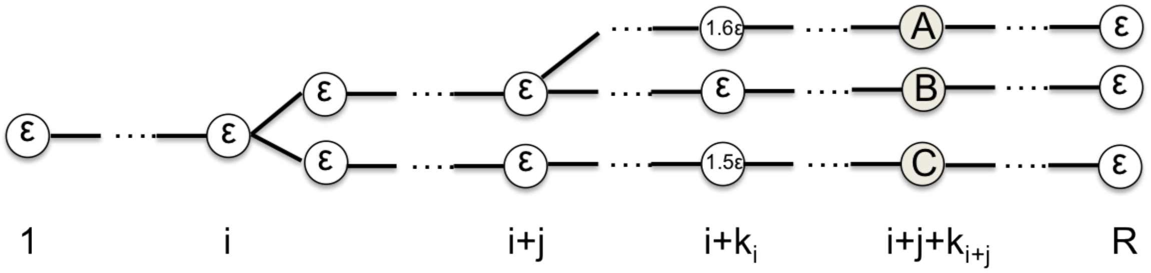


Figure 5.4: Routing game construction for the $j > 0$ case.

rules, by construction A, B, and C are more than m hops from player i. In a PBE with m-local strategies, it is thus necessary for the routing decisions of player i to be independent of the number of coin flips held by players at A, B, and C, that is, for the routing decision to be the same for G'' and G'''.

We show that player i's best response to the equilibrium strategies of the other players depends on G'' or G'''. We first consider game G''. Using backwards induction, note that each player must strictly prefer to route the task forward (to the right) instead of backwards at all times, regardless of the lookahead induced by their routing payment. This is because a player's expected payment is based on the number of coin flips collected and, as before, one can always collect more coin flips in the forward direction (as going backwards necessitates visiting a node that's been visited before). In this case, the top player at $i+j$ would route up because the $i+k_i$-th player would have more coin flips (1.6ϵ) and is within the scope of the routing payment. Given knowledge of the values at A and B, it is thus strictly better for player i to route up in G''.

Consider now game G'''. By backwards induction, each player strictly prefers to route forward because doing so guarantees the largest payment along the way for any lookahead. The top player at $i+j$ will route along the middle path in equilibrium because he would receive $\epsilon + 1.7\epsilon$ from coin flips at the middle path of $i+k_i$ and $i+j+k_{i+j}$ versus the $1.6\epsilon + \epsilon$ along the top path. In this case, player i would rather route down instead of up because it would collect 0.5ϵ more coin flips due to the 1.5ϵ at $i+k_i$ on the bottom path. However, since player i's best response routing decision should be the same for game G'' and G''', we have a contradiction. □

5.6 Simulations and Results

The equilibrium strategies induced by local routing rules can be viewed as providing a heuristic algorithm for computing an optimal route over a network. We now demonstrate via simulations that routing decisions based on local rules can effectively aggregate information as a task is routed through the network.

We consider connected random graphs with 100 nodes and average degree $d \in \{4, 10\}$, generated using the Watts-Strogatz model [100]. By varying the re-wiring probability β, the model allows us to generate graphs that interpolate between a regular lattice ($\beta = 0$) and a $G(n,p)$ random graph ($\beta = 1$), with small-world networks emerging at intermediate values of β. We associate each node with a number of coin flips. Coin flips are drawn independently either discretely from U[1,10] or from a skewed distribution where the value is 1 with probability 0.9 and 46 with probability 0.1. The two distributions have equal mean (5.5), but the skewed distribution more closely resembles a setting where there are few experts. For graphs generated in this manner, we simulate player strategies under local routing rules (MRSR, and m-hop with $m = 2$, $m = 3$) by computing local paths in the manner noted in Theorem 5.4,

		$d = 4$			$d = 10$		
β	Dist.	MRSR	m=2	m=3	MRSR	m=2	m=3
.03	U	69	71	72	83	84	85
0.1	U	71	72	75	85	86	87
1.0	U	76	78	80	89	89	90
.03	S	80	87	104	150	183	227
0.1	S	88	109	146	181	226	259
1.0	S	120	155	183	227	258	278

Table 5.1: Comparison of routing performance based on the average number of coin flips collected after 10 steps. Values represent averages over 100 trials. We considered connected Watts-Strogatz graphs based on uniform (U) and skewed (S) coin flip distributions with fixed mean (5.5). In all cases, routing based on local routing rules collected significantly more coin flips than the 55 coin flips (upper bound) we would expect to collect from routing randomly.

where revisited nodes are treated as having no value. As a baseline, we consider a random routing rule that routes to a random neighbor, and whenever possible, to a random neighbor who has yet to be assigned the task. Note that the expected performance of the baseline is bounded by 5.5 coin flips per round, as we would expect from randomly picking unvisited nodes in the graph.

Table 5.1 shows the average number of coin flips collected after 10 steps by players following local routing rules on graphs with varying β, average degree, and coin flip distribution over 100 trials (standard errors are small and hence not reported). We see that routing rules are particularly effective in cases where there are few experts (S), and when the graph has a sufficiently high connectivity (higher d and β) such that paths exist through which experts can be routed the task. But even in cases with uniformly distributed coin flips (U) and low average degree ($d = 4$), local routing rules collect significantly more coin flips than the upper bound of 55 we would expect

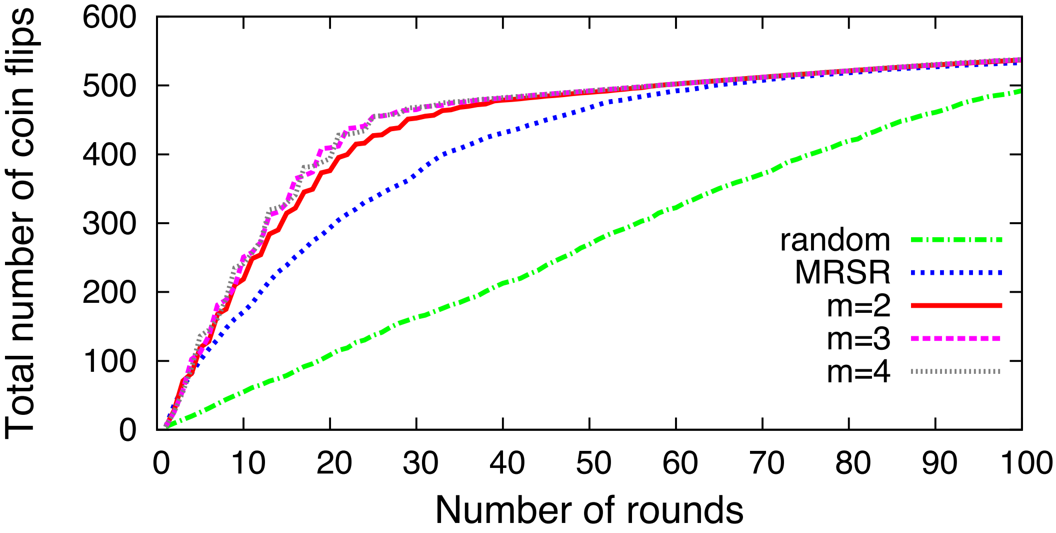


Figure 5.5: Comparison of routing performance based on the average number of coin flips collected for graphs with $\beta = 0.1$, $d = 10$, and skewed coin flip distributions. Values represent averages over 100 trials. Routing based on local routing rules collected significantly more coin flips over fewer rounds than routing based on the random routing rule.

from randomly choosing nodes. Despite connectivity constraints, paths still included many high valued nodes (recall the max per node is 10).

The difference in routing performance among local routing rules is rather small for uniformly distributed values, but is more significant when the distribution is skewed. In this case, effective routing may require finding short paths to experts who are not neighbors. That said, this difference shrinks for graphs with higher degree, as high-value nodes become more reachable (recall that as graphs approach cliques, myopic is optimal).

Figure 5.5 shows the average number of coin flips collected by local routing rules as we progress through the routing game on graphs with $\beta = 0.1$, $d = 10$, and skewed coin flip distributions. We see that routing based on local routing rules collected significantly more information over fewer rounds than routing based on the random routing rule. For $m \geq 2$, the performance under the local routing rules are essentially the same, suggesting that we can sometimes achieve near-optimal performance globally with just two-hop local common knowledge.

With the random routing rule, we see that the rate of information aggregation stays nearly constant throughout the routing game. Since the rule routes to new players whenever possible, this suggests that the graph is well-connected and that new players can often be routed the task even later in the game when many players have already participated. With local routing rules, we see that the rate of information aggregation eventually slows down, which denotes the point at which virtually all experts have been routed the task.

5.7 Discussion

We consider the opportunity for incentivizing the joint refinement and routing of tasks among people within a network, focusing on prediction tasks. We introduce and study local routing rules which, in equilibrium, support people truthfully contributing information and routing tasks based on simple computations that nevertheless lead to effective information aggregation.

In our analysis, we have assumed that bits of signal are conditionally independent. But in some settings, players' signals may be *conditionally dependent*, and accurate predictions may depend on collecting the complementary information held by different players. In this setting, our theoretical results continue to hold with small modifications. First, it is no longer sufficient to maintain a posterior estimate. Instead, we need to explicitly keep track of the information contained in players' signals. Second, we need to restrict players from participating more than once, or alternatively, from being paid for their information beyond the first time. This prevents the type of incentive issues that may occur in prediction markets, in which participants with conditionally dependent signals may be better off withholding some information until complementary information has been reported to the market [14].

While local routing rules enable equilibrium behavior for which the inference required of participants is tractable, these rules still assume that participants are rational in that they maximize their expected payoff. In practice, participants can make mistakes and route suboptimally. But even so, local routing rules may provide for a robust design in which participants are incentivized towards making good decisions even if their decisions are not optimal. Since local routing rules are based on strictly proper scoring rules, which in our setting are *accuracy-rewarding* [51], more accurate predictions will lead to strictly higher payoffs. Furthermore, since the equilibrium is constructed within local paths, any "mistakes" also remain local, and do not affect the routing decisions of later participants outside of local reach.

In crafting local routing rules, we demonstrated a means for designing incentives that explicitly enable players to make simple computations in equilibrium. The key idea is to ensure that players need only make decisions based on information they are guaranteed to have. This requires that players' routing payments are localized and that any chains of reasoning are limited to within local neighborhoods. We believe this idea generalizes beyond prediction tasks and can enable effective solving and routing over social networks in a variety of settings.

There are many possible directions for future work on task routing. One direction is to study routing performance under specialized network topologies and knowledge distributions. Another direction is to extend our models to consider the intrinsic value and cost for solving or routing. In this direction, we are also interested in introducing communication or sensing mechanisms coupled with means of tracking costs of acquiring information, in order to take into account and study the process through which individuals make contributions.

We are interested in developing general principles and methods for solving and routing with humans and machines, and in considering other types of meta-knowledge participants may have about the expertise of others in a social network. In addition to multiple opportunities to address task-level issues, there are also opportunities to address organizational issues related to distributing streams of tasks in a manner that takes into account people's solving and routing abilities over a spectrum of tasks, as well as participants' changing levels of attention, motivation, and availability. We elaborate on this direction in Chapter 9.

Chapter 6

Automated Environment Design

In the previous chapters, we introduced a number of designs for crowdsourcing complex tasks that are effective in recruiting individuals with relevant expertise to join in problem solving and enabling coordination and collaboration. To promote desired behaviors and outcomes, we focused on reasoning about the crowd's abilities, limitations, and work processes in order to construct workflows, interfaces, and incentive mechanisms that are tailored to the characteristics of the crowd.

While we have focused thus far on the design of human computation systems, understanding participants and their behavior is crucial for designing any social or economic system. Participants have varied knowledge and abilities, interests and motivations, availability, and decision-making processes. Together with the decision environment, these elements influence participants' decisions on what actions to take. Designers can draw on what they know, but do not typically have a complete understanding of participants and cannot always predict their behavior. For this reason, solving a computational environment design problem may require experimenting with alternative designs, and iterating to improve designs over time to better promote desired behaviors.

The Internet provides a number of tools for designers that support a data-driven, iterative design process. Frameworks, style sheets, and content management systems make it easier to modify or extend existing designs. Web analytics software tracks individual and group behaviors over time, and provides information on trends and patterns in the data. Tools for A/B testing allow designers to put hypotheses to the test, by measuring the performance of competing designs against defined objectives.

But despite having a rich set of tools, the process of discovering effective designs is still largely manual, tedious and ad hoc. Designers spend significant time and effort coming up with alternative designs, that may consist of small modifications geared towards making immediate improvements. Without particular regard to gaining a deeper understanding of participants or of potential interactions among design elements, this may lead to an experimentation process that tries to hill-climb toward a solution at a local, rather than global, maximum. Designers may miss out on parts of the design space where better solutions exist, and ultimately fail to promote desired behaviors and outcomes.

A more principled and automated approach to experimentation may lead to more effective designs more quickly, while requiring less manual effort. Such an approach may use observations of participant behaviors to not only evaluate competing designs, but also to refine our understanding of participants' abilities, motivations, and decision-making processes. This knowledge may allow us to reason about the design space more globally, and to discover designs that we would otherwise have missed. To reduce the amount of manual effort required, and to discover effective designs more quickly, automated procedures can be employed to seamlessly combine domain knowledge with machine-driven processes that optimize the choice of experiments and refine existing models based on observed behavior. From the perspective of the designer, an automated system may simply take as input a set of available interventions, the objective of the designer, and a model of participants, and provide as output an intervention that promotes actions and outcomes meeting the objective whenever such interventions exist, or otherwise learn something new about participants.

In this chapter, we introduce a general approach for *automated environment design.* Section 6.1 presents a formal model of the automated environment design problem. Section 6.2 provides an *active, indirect elicitation* framework that automatically drives an objective-oriented, iterative design process in which a system indirectly learns about participants based on observations of participant behavior in response to experiments chosen based on current knowledge. Section 6.3 introduces the problem of *policy teaching* as a case study, in which an interested party aims to provide limited rewards to induce an agent in a sequential-decision setting to follow a desired policy. We construct an active, indirect elicitation algorithm, that without prior knowledge of the agent's reward function, is guaranteed to discover rewards in a constrained reward space that elicit the desired policy after few interactions, as long as such rewards exist. Section 6.4 describes how our methods and results may generalize to other automated environment design problems, and discusses our assumptions as well as alternative models and approaches for automated environment design.

6.1 Model for Automated Environment Design

We consider situations in which an automated system, which we refer to as an *interested party*, seeks to design or modify aspects of a social or economic system on the Internet with the intent of eliciting desired actions and outcomes. For simplicity of notation and without loss of generality, we model participants in a system as if they were a single *agent*.[1] A model for an automated environment design problem consists of a decision environment, an agent model, an environment change, an admissibility condition, an environment transition function, and a goal function. Below we define these components, and present static and dynamic formulations of the problem.

Consider an agent who acts in a *decision environment* $e \in \mathcal{E}$ based on his agent model $\mathcal{M} = \{\theta, f, \Lambda\}$, which consists of the *model parameters* $\theta \in \mathcal{I}$; the *agent function* $f : \mathcal{I} \times \mathcal{E} \rightarrow 2^{\mathcal{X}}$, where $\mathcal{X}$ is the decision space; and the *actuation function* $\Lambda : \mathcal{X} \times \mathcal{E} \rightarrow \mathcal{O}$, where O is the output space. The model parameters represent the agent's preferences and capabilities, and contains information private to the agent. The agent function takes the model parameters and environment as input and identifies (perhaps multiple, equivalent) decisions, which describe how the agent plans to act in the environment. The actuation function takes the agent's decision and the environment and provides an output representing the agent's actual actions in the environment. Although described here as deterministic for expositional clarity, the actuation function need not in general map a decision to an output deterministically, and may instead sample from a distribution over actions. Furthermore, while

[1] Interactions among participants can be captured by having the agent model take into account how participants in a decision environment may interact and make decisions based on other participants' actions.

the agent's actions may sometimes reveal the agent's exact decision, we assume that decisions are not directly observable.

We make a couple of assumptions about the agent model. First, we assume that the agent fully perceives the decision environment and makes decisions with respect to that knowledge.[2] Second, we assume that f and Λ are fixed and known to the interested party. This abstraction implies that if the interested party had full knowledge of the agent's model parameters, he would be able to predict the agent's decisions and a distribution over agent actions in the designed environment. Third, we assume the agent can compute f on any input he encounters, such that any computational limitations of the agent is embedded within f. Lastly, we assume that the agent makes a single decision $x \in f(\theta, e)$ when f returns a non-singleton set of decisions, with this tie-breaking rule *a priori* unknown to the interested party.

Having described the agent model, we turn to consider the interested party's problem. We assume the presence of a *base environment* e^0, which the interested party can modify via an *environment change* $\Delta \in \boldsymbol{\Delta}$. The *environment transition function* $\mathcal{F} : \mathcal{E} \times \boldsymbol{\Delta} \rightarrow \mathcal{E}$ takes the base environment e^0 and an environment change as input and outputs a modified environment. We assume this function is deterministic and known to the interested party. Furthermore, we assume that once the environment is modified, the agent acts with respect to a decision in the modified environment. Since the environment enters as input into the agent function, modifying the environment may influence the agent's decision and actions. We assume that the interested party fully perceives the environment, and can observe the agent's actions.

[2] Alternatively, one can define f based on the agent's perceptual inputs as opposed to the environment. For sake of exposition we do not explicitly model the agent's perception.

We assume that the agent is myopic with respect to environment changes. That is, the agent follows his agent function and does not reason about future changes to the environment when making current decisions. This seems reasonable in social and economic systems on the Web, in which there are large numbers of users, most of whom tend to use services as desired without reasoning about how systems may change in the future. Furthermore, as design decisions tend to be guided by the behaviors of many users, a single individual's actions are unlikely to affect a system's (re)design. That said, it is generally possible for users to take actions with the intent of influencing environment changes; we elaborate on this issue later in the chapter.

Given a set $X \in 2^{\mathcal{X}}$ of agent decisions that may result from an environment change, the admissible set $\mathit{admissible}(X) \subseteq \mathbf{\Delta}$ characterizes the space of allowable environment changes. Admissibility conditions can model the interested party's design costs and constraints, both of which may potentially depend on the agent's decisions. For example, an environment change that rewards user actions may be infeasible if agent decisions in the modified environment lead to actions that require the interested party to issue more rewards than he has available. We assume the admissible set always contains a null element Φ, corresponding to no environment change.

Finally, we define the goal of the interested party. The *goal function* $\mathcal{G} : \mathcal{X} \times \mathbf{\Delta} \times \mathcal{I} \times \mathcal{E} \to \Re$ takes the agent's decision under the modified environment, the environment change, the agent's model parameters, and the modified environment as input and outputs the value to the interested party.[3] The goal may depend on

[3] Since the agent's decision is not directly observable, in practice an interested party may use samples of observed actions to evaluate admissibility and goal conditions. Since the agent's model parameters are also private to the agent, the interested party may need to evaluate the goal function with respect to beliefs about the actual model parameters.

Environment	a Web 2.0 site
Agent model parameters	preferences over site modules; time available to spend online
Agent function	decision on what to do on the site based on interest and availability
Actuation function	actual user actions on the site based on user decisions
Environment change	adding, removing, and moving modules in the user interface
Admissibility condition	limit to changes within template; keep main components centered and visible
Environment transition function	describes how the user interface changes
Goal function	retention rate among new users; the volume of content contributed

Table 6.1: An example showing the various components of a computational environment design problem in which an interested party wishes to redesign the user interface of a Web 2.0 site to improve retention rate and increase the volume of user contributions.

(a) the agent's decision because it determines (the distribution over) agent actions and outcomes; (b) the environment change because this may come at a cost; (c) the model parameters because the designer may wish to consider the value to the agent; and (d) the modified environment because the interested party may value the agent's decisions differently in different environments.

To get a sense of how the model applies to computational environment design problems we may encounter in practice, see Table 6.1, which illustrates the various components of a computational environment design problem in which an interested party wishes to design the user interface of a Web 2.0 site to improve retention and increase the volume of user contributions.

6.1.1 Static Formulation

As a special case, we first present the *static formulation* of the automated environment design problem, in which we assume that the agent's model parameters are known to the interested party. The goal is to find an admissible Δ such that the agent's elicited behavior in the modified environment maximizes the goal function $\mathcal{G}$. Since the interested party already knows the agent function and the environment, we can think of the interested party's problem as a one-shot optimization problem.

In the case of multiple possible decisions in the range of the agent function, the agent may not select the one desired by the interested party. To be certain that the agent selects decisions desired by the interested party, our formulation assumes that the agent selects the worst possible decision for the interested party's goal function:

Definition 6.1. *Given an environment e, the static computational environment design problem is an optimization problem to find an environment change Δ that maximizes the interested party's goal function in the worst case:*

$$\max_{\Delta}[\min_{x_T} \mathcal{G}(x_T, \Delta, \theta, e')] \tag{6.1a}$$

$$\textit{subject to:} \quad e' = \mathcal{F}(e, \Delta) \tag{6.1b}$$

$$x_T \in f(\theta, e') \tag{6.1c}$$

$$\Delta \in \textit{admissible}(f(\theta, e')) \tag{6.1d}$$

In the case that the agent function outputs singleton decision sets, the objective of the optimization simplifies to $\max_{\Delta} \mathcal{G}(x_T, \Delta, \theta, e')$.

The constraints ensure that e' is the modified environment (6.1b), that the model parameters and modified environment induce some decision x_T (6.1c), and that the environment change is admissible with respect to all possible agent decisions (6.1d) consistent with the new environment.

6.1.2 The Dynamic Formulation

In the more interesting case, and the focus of this chapter, the agent's model parameters will initially be, at least partially, unknown to the interested party. Since the agent function depends on both the environment and the model parameters, the interested party may not be able to immediately identify admissible environment changes that promote the desired behavior. To address this, the interested party can experiment with alternative designs and have repeated interactions with the agent. In each interaction, the interested party can modify the environment and observe the agent's actions in the modified environment.

Observations and measurements can inform which experiments to conduct in subsequent interactions, and the goal is to arrive at effective designs quickly. An example objective may be to induce desired decisions after few interactions, without being concerned about the cost of experimentation. Given a target goal value $\underline{\mathcal{G}}$, we can represent this objective as minimizing the number of rounds until we find an admissible Δ that induces a decision environment e' in which the agent's decision x_T satisfies $\mathcal{G}(x_T, \Delta, \theta, e') \geq \underline{\mathcal{G}}$.

More generally, we can imagine that in the midst of experimentation, the interested party is (in a separate process) using the results of experimentation to deploy environment changes. Deployed designs may be designs from past experiments or new designs that are computed using currently available information. Viewed this way, the interested party may wish to maximize one of several objectives that represent the exploration and exploitation tradeoff of having effective designs to deploy now versus later. For any point along this spectrum, the goal is to design experiments that maximize some measure of the expected goal value derived from deploying environment changes now and in the future. Different objectives weigh the value derived from experimentation differently, depending on when particular designs are discovered and deployed.

6.2 An Active, Indirect Elicitation Framework

Solving the dynamic formulation requires discovering effective designs quickly. To make efficient use of experiments, we can draw on observations and measurements to not only evaluate competing designs, but to refine our understanding of model parameters guiding the agent's behavior. For example, one can infer from observing consumer purchases and worker performance on tasks information about the underlying preferences and abilities that guide the person's decisions and actions. As an agent makes decisions in different environments with respect to his *actual* model parameters, we can use observed behavior to make inferences about the space of model parameters consistent with observations. Even without identifying the agent's actual model parameters, such information and knowledge may allow us to better predict how an agent will respond to different designs. This enables us to reason more effectively about the design space.

Taking advantage of this insight, we introduce an *active, indirect elicitation framework* that drives an automated, iterative design process that interleaves optimiza-

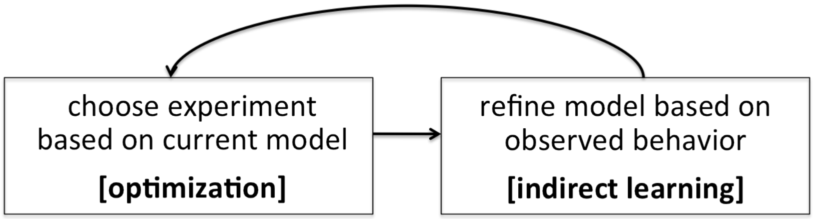

choose experiment based on current model
[optimization]
refine model based on observed behavior
[indirect learning]

Figure 6.1: The active, indirect elicitation framework combines optimizing experiments based on current knowledge of model parameters with indirect learning of model parameters based on observed behavior.

tion of appropriate experiments with indirect learning of model parameters (see Figure 6.1). In each round, an experiment is designed using knowledge of the agent's model parameters, and seeks to derive new information from observing potential agent actions in the modified environment. Following an interaction, the knowledge of model parameters is refined by making inferences based on observed behavior. Since the goal is ultimately to elicit desired actions, experiments should be selected with the interested party's objective in mind, and not just for the sake of learning about the agent's underlying model parameters.

An algorithm based on the active, indirect elicitation framework contains two components: an inference procedure and an elicitation strategy. An *inference procedure* updates the interested party's beliefs about the actual model parameters, by incorporating new observations from experiments. Let $\mathcal{H}$ denote the history of past elicitation rounds, such that $(o^t, e^t) \in \mathcal{H}$ denotes observed actions o^t in environment e^t in round t. For all observations $(o^t, e^t) \in \mathcal{H}$, the agent's actual model parameters θ^* must satisfy $f(\theta^*, e^t) = x^t$, where x^t is the agent's decision in round t that, through the actuation function $\Lambda(x^t, e^t)$, led to the observed output o^t. By making inferences based on the relationships among these components, the inference procedure allows us to refine our beliefs about θ^* over time. Indirectly, this enables us to better predict the agent's decisions and actions in response to different environment changes.

The *elicitation strategy* optimizes for experiments based on our beliefs, as provided by the inference procedure using the history $\mathcal{H}$. Depending on the interested party's objective, the elicitation strategy may focus on obtaining information that would most immediately lead to an improved design, or be more forward looking by taking into consideration the potential value that can be derived in the future from information learned now.

6.3 Case Study: Policy Teaching

For an algorithm based on the active, indirect elicitation framework to be practically useful, the inference procedure and elicitation function must be computationally tractable and help to discover effective designs quickly. To illustrate how the active, indirect elicitation framework can be applied to a specific automated environment design problem, we consider as a case study the problem of *policy teaching*.

Policy teaching considers a Markov Decision Process (MDP) setting in which an interested party can associate rewards with world states to affect an agent's policy. The interested party can observe the agent's decisions in response to provided incentives, but generally does not know the agent's reward function. The interested party can interact multiple times with the agent, but cannot directly impose actions on the agent. The goal of the interested party is to quickly identify feasible incentives (i.e., rewards from a constrained reward space) that induce the agent to follow a desired behavior or policy, when this is possible.

Policy teaching models situations on the Web in which an interested party can modulate costs and rewards in attempt to elicit desired actions. For example, a retailer such as Amazon may want customers to make frequent purchases and write product reviews, and may be willing to provide discounts on products and recognize top reviewers. Question-and-answer sites such as Yahoo! Answers and Stack Overflow may want users to answer lingering questions and generally spend time on the site, and can tweak their interfaces to make it easier to contribute (thus reducing the cost of effort) and offer points and badges as social rewards. Ad networks such as Google AdSense may want publishers to design their web sites to facilitate effective advertising, and can offer a share of the ad revenue to entice a publisher to choose a particular web layout.

We focus on the policy teaching problem in which the goal is to induce a fixed, *prespecified desired policy*. Section 6.3.1 provides a model of this automated environment design problem. Section 6.3.2 shows that in the static case, the problem can be formulated as a linear program. Section 6.3.3 considers the more likely case where the agent's reward function is unknown, and introduces an active, indirect elicitation algorithm that is guaranteed to converge after a few rounds to discover rewards to apply to states that induce the desired policy. To make the algorithm tractable, we apply results from sampling in convex spaces [6] to arrive at a polynomial time algorithm that maintains the same convergence guarantees with arbitrarily high probability. Section 6.3.4 summarizes our results and discusses a few extensions.

6.3.1 Model

The policy teaching problem considers an agent performing a sequential decision task with respect to an infinite horizon MDP $M = \{S, A, R, P, \gamma\}$, where S is a finite set of states, A is a finite set of possible actions, $R : S \rightarrow \Re$ is the reward function, $P : S \times A \times S \rightarrow [0, 1]$ is the transition function, and $\gamma \in (0, 1)$ is the discount factor. Given M, the agent's decision problem is to choose actions for each state to maximize the expected sum of discounted rewards. Let π denote a stationary policy, such that $\pi(s)$ is the action the agent executes in state s. Given a policy π, the value function $V^{\pi}(s) = R(s) + \gamma \sum_{s' \in S} P(s, \pi(s), s')V^{\pi}(s')$ captures the expected sum of discounted rewards from state s. Similarly, the Q function captures the value of taking an action a and following the policy π in future states, such that $Q^{\pi}(s, a) = R(s) + \gamma \sum_{s' \in S} P(s, a, s')V^{\pi}(s')$. By Bellman optimality [76], an optimal policy π^* maximizes the Q function in every state, such that $\pi^*(s) \in \arg\max_{a \in A} Q^{\pi^*}(s, a)$. We assume the agent can compute an optimal policy of his MDP, and that his inherent reward function R is persistent.[4]

We consider an interested party whose goal is to induce a prespecified target policy π_T. The interested party knows S, A, P, and γ, but not the agent's reward function R. We assume that the interested party can observe the agent's actions, and that observed actions completely reveal the agent's policy (decision). The interested party can influence the agent's reward function by providing incentives $\Delta : S \rightarrow \Re$. We assume that Δ affects the agent's reward function linearly, such that the agent plans

[4] Mapping back to the general model, the agent function in this setting forms the agent's decision by computing the optimal policy with respect to the MDP model M, which captures aspects of both the environment and the agent's model parameters.

with respect to $M' = \{S, A, R + \Delta, P, \gamma\}$ in the modified environment. Following our base assumption that the agent is myopic with respect to environment changes, we assume the agent is *myopically rational* and follows the optimal policy in the modified environment.

To capture the idea that the interested party may only be able to provide limited incentives, we define a notion of admissibility:[5]

Definition 6.2. *An incentive function $\Delta : S \rightarrow \Re$ is admissible given budget D_{max} and Δ_{max} with respect to a policy π_T if it satisfies the following linear constraints, denoted $\Delta \in admissible(\pi_T)$:*

$$V_{\Delta}^{\pi_T}(s) = \Delta(s) + \gamma P_{s,\pi_T(s)} V_{\Delta}^{\pi_T}, \forall s \in S \qquad \textit{Incentive value.} \tag{6.2}$$

$$V_{\Delta}^{\pi_T}(start) \leq D_{max} \qquad \textit{Limited spending.} \tag{6.3}$$

$$0 \leq \Delta(s) \leq \Delta_{max}, \forall s \in S \qquad \textit{No punishments.} \tag{6.4}$$

The incentive value $V_{\Delta}^{\pi_T}(s)$ in Definition 6.2 captures the total sum of expected discounted incentives provided to an agent following policy π_T starting from state s. The limited spending constraint limits the total incentives provided to D_{max} when the agent performs π_T from the *start* state.[6] The "no punishment" condition ensures that only bounded, positive incentives are provided, which seems quite fitting in many of the web domains that motivate this work.[7] We focus primarily on finding admissible

[5] The general model allows admissibility conditions to be defined over a set of decisions, but here we define it with respect to a single decision π_T. Given that the interested party's goal is to induce a single target policy, it is reasonable to assume that he would only be interested in discovering and deploying incentives Δ that strictly induce π_T and are admissible with respect to π_T.

[6] The use of a single start state is without loss of generality, since it can be a dummy state whose transitions represent a distribution over possible start states.

[7] Alternative definitions of admissibility are possible as well. Our methods are not specific to a particular admissibility definition, so we will not pursue the issue further.

incentives to elicit the desired policy quickly, and only consider minimizing cost as a secondary objective.

6.3.2 The Known Rewards Case

To develop intuition, we first consider the static formulation in which the interested party knows the agent's reward function. The policy teaching problem is to find minimal admissible incentives that induce the desired policy π_T. To capture the space of rewards that are consistent with a particular policy, we first define the concept of *inverse reinforcement learning* (IRL) [68]:

Definition 6.3. *Given a policy π and $M_{-R} = \{S, A, P, \gamma\}$, let $\{R : R \in IRL^{\pi}\}$ denote the set of reward functions for which π is optimal for the MDP $M = \{S, A, R, P, \gamma\}$. Furthermore, for $\epsilon > 0$, let $\{R : R \in IRL^{\pi}_{\epsilon}\}$ denote the set of rewards for which π is uniquely optimal for M by a slack of at least ϵ, such that $Q^{\pi}(s, \pi(s)) - Q^{\pi}(s, a) \geq \epsilon$ for all $s \in S$, $a \in A \backslash \pi(s)$.*

The policy teaching problem then aims to find incentives leading to a reward function that is consistent with the desired policy:

Definition 6.4. ***Policy teaching with known rewards.*** *Given an agent MDP $M = \{S, A, R, P, \gamma\}$, target policy π_T, incentive limits D_{max} and Δ_{max}, and $\epsilon > 0$, if there exists admissible Δ such that $(R + \Delta) \in IRL^{\pi_T}_{\epsilon}$, find such a Δ to minimize $V^{\pi_T}_{\Delta}(start)$.*

The definition requires that the provided incentives strictly induce the desired policy. This avoids scenarios in which an agent is indifferent among multiple optimal policies and may choose a policy other than that which is desired by the interested party.

To solve this problem, we need to (1) locate the space of reward functions under which π_T is uniquely optimal and (2) find an admissible incentive Δ that maps the agent's reward into this space. We apply a well-known result from inverse reinforcement learning, which shows that the space of rewards consistent with a particular (uniquely) optimal policy is given by a set of linear constraints:

Theorem 6.1. (Ng and Russell [68]) *Given a policy π written as $\pi(s) \equiv a_1$ and $M_{-R} = \{S, A, P, \gamma\}$, $R \in IRL^{\pi}$ satisfies:*

$$(\mathbf{P_{a_1}} - \mathbf{P_a})(\mathbf{I} - \gamma \mathbf{P_{a_1}})^{\mathbf{-1}}\mathbf{R} \succeq \mathbf{0} \quad \forall a \in A \backslash a_1 \tag{6.5}$$

Furthermore, for $\epsilon > 0$, $R \in IRL^{\pi}_{\epsilon}$ satisfies:

$$(\mathbf{P_{a_1}} - \mathbf{P_a})(\mathbf{I} - \gamma \mathbf{P_{a_1}})^{\mathbf{-1}}\mathbf{R} \succeq \epsilon \quad \forall a \in A \backslash a_1 \tag{6.6}$$

where $\mathbf{P_a}$ is the transition function with respect to action a written in matrix form, $\mathbf{R}$ is the reward function written in matrix form, and $\mathbf{I}$ is the identity matrix.

This theorem leads directly to our first result:

Theorem 6.2. *The following linear program solves policy teaching with known rewards:*

$$\min_{\Delta} V_{\Delta}^{\pi_T}(start) \tag{6.7}$$

$$R_T(s) - \Delta(s) = R(s) \qquad \forall s \tag{6.8}$$

$$((\mathbf{P_{a_1}} - \mathbf{P_a})(\mathbf{I} - \gamma \mathbf{P_{a_1}})^{\mathbf{-1}}\mathbf{R_T})[\mathbf{s}] \succeq \epsilon \qquad \forall s, a \in A \backslash a_1 \tag{6.9}$$

$$\Delta \in admissible(\pi_T) \tag{6.10}$$

where $a_1 \equiv \pi_T(s)$ denotes the actions of the target policy, $\mathbf{P_a}$ is the transition function with respect to action a written in matrix form, and $\mathbf{R_T}$ is a reward function that strictly induces π_T written in matrix form.

6.3.3 The Unknown Rewards Case

In most situations, the interested party will not know the agent's reward function. This leads to the following problem definition:

Definition 6.5. ***Policy teaching with unknown agent reward.*** *Consider an agent following a policy π with respect to an MDP $M = \{S, A, R, P, \gamma\}$. An interested party observes the agent's policy, and knows $M_{-R} = \{S, A, P, \gamma\}$ but not R. Given target policy π_T, incentive limits D_{max} and Δ_{max}, and $\epsilon > 0$, if there exists an admissible Δ for which $(R + \Delta) \in IRL^{\pi_T}_{\epsilon}$, find an admissible Δ and observe agent policy π' such that $\pi' = \pi_T$ after few interactions.*

We assume that direct queries about the agent's preferences are unavailable and that preference information must be inferred from observations of agent behavior. This is often true on the Web. While firms such as Amazon and Facebook can observe user actions, it may be considered intrusive for them to directly ask their users for preference information. Doing so may disrupt from the user experience, and users may question their motives.

We develop an algorithm based on the active, indirect elicitation framework, wherein the space of potential agent rewards is narrowed by drawing additional IRL constraints based on observations of agent behavior in response to provided incentives. We assume the agent's reward function is bounded in absolute value by R_{max}

in every state. Within these bounds, we maintain an "IRL space" of reward functions that are consistent with observations *and* that have associated admissible incentive functions that can strictly induce the desired policy with some minimal slack $\epsilon > 0$.

At every iteration, the *elicitation function* makes a guess $\widehat{R}$ at the agent's true reward by choosing a point in the IRL space. If the guess is correct, providing the associated incentives $\widehat{\Delta}$ will strictly induce π_T. If instead the agent performs a policy $\pi' \neq \pi_T$, we know that $\widehat{R}$ must not be the agent's true reward R. Furthermore, we know that $R+\widehat{\Delta}$ induces π', which allows the *inference procedure* to add the following IRL constraints to the IRL space:

$$(\mathbf{P_{a_1}} - \mathbf{P_a})(\mathbf{I} - \gamma \mathbf{P_{a_1}})^{-\mathbf{1}}(\mathbf{R} + \widehat{\boldsymbol{\Delta}}) \succeq \mathbf{0} \quad \forall a \in A \backslash a_1 \tag{6.11}$$

where $a_1 \equiv \pi'(s)$ denotes the actions of the observed policy, $\mathbf{P_a}$ is the transition function with respect to action a written in matrix form, $\widehat{\Delta}$ is the incentive provided, and $\mathbf{R}$ is the agent's reward function written in matrix form.

IRL constraints contain $|S||A|$ constraints on R and restrict the space of possible rewards to the intersection of the previous IRL space and the convex polytope implied by the added constraints. Since we are only interested in the agent's reward for the purpose of solving the policy teaching problem, we can stop the elicitation process as soon as we observe the desired policy or as soon as the IRL space becomes empty (declaring the problem impossible).

We use the following notation. All constraints are added to a constraint set K, such that instantiations of variables satisfy all constraints in K. An instantiation of a variable R is denoted as $\widehat{R}$. Algorithm 6.1 gives the elicitation method.

Algorithm 6.1 Active indirect elicitation for policy teaching

```
Consider agent policy $\pi$, desired policy $\pi_T$, $\epsilon > 0$
Variables $R$, $R_T$, $\Delta$; constraint set $K = \emptyset$
Add $R \in \text{IRL}^{\pi}$, $|R(s)| \leq R_{max}\ \forall s \in S$ to $K$
Add $R_T \in \text{IRL}_{\epsilon}^{\pi_T}$, $\Delta = R_T - R$ to $K$
Add $\Delta \in admissible(\pi_T)$ to $K$
loop
    Find $\widehat{\Delta}$, $\widehat{R}$, $\widehat{R_T}$ satisfying all constraints in $K$
    if no such values exist then
        return FAILURE
    else
        Provide agent with incentive $\widehat{\Delta}$
        Observe $\pi'$
        if $\pi' = \pi_T$ then
            return $\widehat{\Delta}$
        else
            Add $(R + \widehat{\Delta}) \in \text{IRL}^{\pi'}$ to $K$
```

Theorem 6.3. *Algorithm 6.1 terminates in a finite number of steps with a solution to the policy teaching problem with unknown rewards or returns FAILURE if no solution exists, regardless of the elicitation function's choice of $\widehat{R}$ and $\widehat{\Delta}$ from K.*

Proof. (sketch) The minimal slack ϵ over the target policy ensures that all points within a closed hypercube of side length $\delta = \frac{\epsilon(1-\gamma)}{\gamma} - \kappa$ centered at $\widehat{R}$ are eliminated by IRL constraints whenever π_T is not observed, for some arbitrarily small $\kappa > 0$.[8] Since the true reward is consistent with IRL constraints, by a pigeonhole argument, only a finite number of such hypercubes of eliminated points can fit in the IRL space before elicitation converges. □

While convergence is a desirable property, in practice the algorithm is only useful if it can induce the desired policy after few interactions. We develop an elicitation strategy that guarantees fast convergence and can be computed tractably.

A Centroid-based Approach

Consider the IRL space at any round of the elicitation process. Since this set of reward functions is characterized by linear constraints, it is convex. We can apply the following result on cutting convex sets:

Theorem 6.4. (Grünbaum [28]) *Any halfspace containing the centroid of a convex set in $\Re^{|S|}$ contains at least $\frac{1}{e}$ of its volume.*

By choosing the centroid of the IRL space of rewards for $\widehat{R}$, any added IRL constraint will cut off at least a constant fraction of the IRL space's volume:

Lemma 6.1. *Let B_K^t denote the IRL space of reward functions implied by the constraints in K before the t-th iteration of Algorithm 6.1. Let c_t denote the centroid of B_K^t. Consider an elicitation strategy that picks $\widehat{R} = c_t$ and any corresponding admissible $\widehat{\Delta}$ for which $(\widehat{R} + \widehat{\Delta}) \in IRL_\epsilon^{\pi_T}$. Providing $\widehat{\Delta}$ will either induce π_T, or lead to adding IRL constraints that eliminate at least $\frac{1}{e}$ of the volume of B_K^t, such that $vol(B_K^{t+1}) \leq (1 - \frac{1}{e})vol(B_K^t)$.*

[8] Throughout this section, a hypercube refers to a closed, axis-aligned hypercube.

Lemma 6.1 implies that after a number of iterations logarithmic in the volume of the IRL space, this volume can be made arbitrarily small. If we can provide conditions under which the desired policy is elicited before the volume of the IRL space falls below some threshold, we can guarantee logarithmic convergence.

One condition that leads to logarithmic convergence is to ensure that all points within a small hypercube centered at the true reward are contained in the initial IRL space and never removed by added IRL constraints (in cases where a solution exist). If points within this hypercube are chosen for $\widehat{R}$, the minimal slack over the target policy ensures that π_T is elicited. Assuming this condition is satisfied, we can stop the elicitation process after logarithmic rounds because we will either elicit the desired policy before the volume of the IRL space drops below the volume of the hypercube, or discover that the true agent reward must not be contained in the initial IRL space and thus there are no possible solutions.[9]

Unfortunately, Algorithm 6.1 may not satisfy this condition because IRL constraints may potentially eliminate some points in the small hypercube centered at the true reward R_{true}. For a reward guess $\widehat{R}$ and associated incentive $\widehat{\Delta}$ that does not induce the target policy, the observed policy π' will be optimal for R_{true} but need not be optimal for all reward functions in the hypercube centered at R_{true}.

Nevertheless, we can modify our current algorithm to ensure that a hypercube of points centered at R_{true} is never eliminated. Since Theorem 6.3 ensures that all points within a closed hypercube of side length δ centered at $\widehat{R}$ are eliminated by added IRL constraints, by convexity there exists a separating hyperplane between

[9] Bertsimas and Vempala [6] used this general observation to formulate an algorithm for finding a point in a convex set specified by a separation oracle with logarithmic queries.

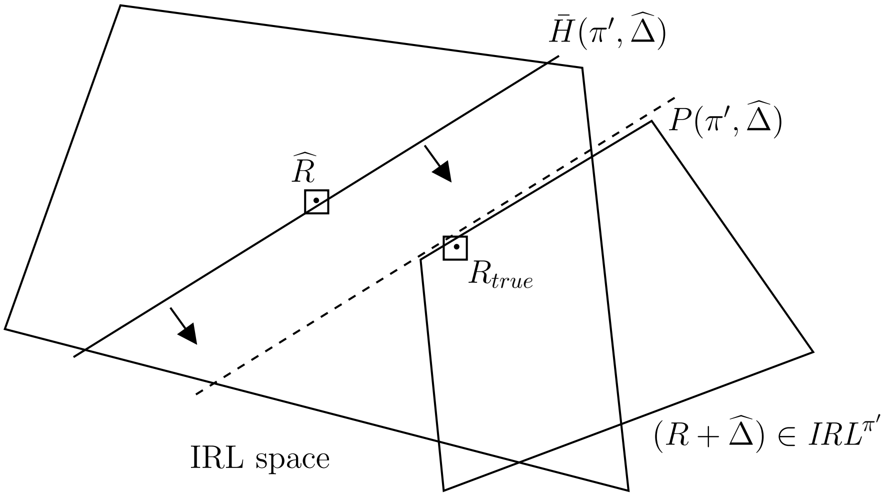

$\bar{H}(\pi', \widehat{\Delta})$
$P(\pi', \widehat{\Delta})$
$\widehat{R}$
R_{true}
IRL space
$(R + \widehat{\Delta}) \in IRL^{\pi'}$

Figure 6.2: A condition that ensures logarithmic convergence requires maintaining a hypercube of points around the true reward R_{true} throughout the elicitation process. The larger polyhedron in the figure represents the IRL space of rewards that have yet to be falsified. Given an observation π' based on incentives $\widehat{\Delta}$, the IRL constraints $(R + \widehat{\Delta}) \in IRL^{\pi'}$ represented by the smaller polyhedron may eliminate some points within the hypercube of points centered at R_{true}. To avoid this, we find a separating hyperplane $P(\pi', \widehat{\Delta})$ between the hypercube centered at $\widehat{R}$ and the IRL constraints, and shift $P(\pi', \widehat{\Delta})$ towards $\widehat{R}$ until it is arbitrarily close to $\widehat{R}$. The resulting hyperplane $\bar{P}(\pi', \widehat{\Delta})$ separates $\widehat{R}$ and the hypercube centered at R_{true}. Adding the corresponding halfspace $\bar{H}(\pi', \widehat{\Delta})$ instead of the IRL constraints ensures logarithmic convergence.

this hypercube and the IRL constraints. Following Figure 6.2, let $P(\pi', \widehat{\Delta})$ be such a separating hyperplane, and let $\bar{P}(\pi', \widehat{\Delta})$ denote a hyperplane that results from relaxing $P(\pi', \widehat{\Delta})$ in the direction perpendicular to itself until it is arbitrarily close to $\widehat{R}$. Let $\bar{H}(\pi', \widehat{\Delta})$ be the halfspace not containing $\widehat{R}$ that is defined by $\bar{P}(\pi', \widehat{\Delta})$. Since $P(\pi', \widehat{\Delta})$ separates R_{true} from a hypercube of side length δ centered at $\widehat{R}$, $\bar{P}(\pi', \widehat{\Delta})$ will separate $\widehat{R}$ from a hypercube of side length δ centered at R_{true}. This ensures that finding $\bar{H}(\pi', \widehat{\Delta})$ and adding it instead of IRL constraints is a sufficient condition for guaranteeing logarithmic convergence.

Since the hypercube of points centered at $\widehat{R}$ and the IRL constraints are both characterized by linear constraints, we can find the separating hyperplane $P(\pi', \widehat{\Delta})$ by solving a simple linear program (e.g., see Theorem 10.4 in Vanderbei [93]). We can easily find $\bar{P}(\pi', \widehat{\Delta})$ by relaxing $P(\pi', \widehat{\Delta})$ until it almost passes through $\widehat{R}$, and define $\bar{H}(\pi', \widehat{\Delta})$ accordingly.

We define a modified version of Algorithm 6.1, denoted Algorithm 6.1*, where: (i) line 3 of Algorithm 6.1* adds $\bar{H}(\pi', \Phi)$ instead of $R \in IRL^{\pi}$ to K (where Φ corresponds to no environment change), (ii) Algorithm 6.1* returns FAILURE if it has not returned after $1 + |S|\lceil \log_b \lceil \frac{R_{max}}{\delta} \rceil \rceil$ rounds, where $b = \frac{1}{1-k}$ for some k such that $0 < k < \frac{1}{e}$, and (iii) given observed policy π' based on $\widehat{\Delta}$, Algorithm 6.1* does not add $(R + \widehat{\Delta}) \in IRL^{\pi'}$ to K and instead finds $\bar{H}(\pi', \widehat{\Delta})$ and add it to K.

Theorem 6.5. *Assume the agent's true reward is bounded by $R_{max} - \delta$ in every state, where $\delta = \frac{\epsilon(1-\gamma)}{\gamma} - \kappa$ for some arbitrarily small $\kappa > 0$. Let B_K^t denote the IRL space of reward functions implied by the constraints in K before the t-th iteration of Algorithm 6.1*, and let $b = \frac{1}{1-k}$ for some k such that $0 < k < \frac{1}{e}$. For any elicitation strategy that picks the centroid of B_K^t for $\widehat{R}$, Algorithm 6.1* terminates with a solution to the policy teaching problem with unknown rewards or returns FAILURE if no solution exists after at most $1 + \lceil \log_b \lceil (\frac{R_{max}}{\delta})^{|S|} \rceil \rceil$ iterations.*

Since the modifications to the algorithm allow us to eliminate the centroid of the IRL space while preserving a closed hypercube of points centered at the agent's true reward, the condition required for logarithmic convergence is satisfied and Theorem 6.5 follows. Here $(\frac{R_{max}}{\delta})^{|S|}$ is the number of non-overlapping hypercubes with side length δ that fit within the bounded space of rewards considered. This can be viewed as the size of the elicitation problem, and the bound given by Theorem 6.5 is logarithmic in this dimension. This logarithmic bound is still linear in the number of states though, because only one of the constraints added at each iteration is guaranteed to cut off a constant fraction of the volume.

Although computing the centroid exactly is #P-hard [77], polynomial time, randomized algorithms exist and extend Grünbaum's result to the case of the approximate centroid. Bertsimas and Vempala [6] showed that any halfspace containing the average of $O(n)$ uniform samples from a convex set in $\Re^n$ will cut off a constant fraction of its volume with arbitrarily high probability. Using this result, we can construct an elicitation strategy that allows $\widehat{R}$ to be computed in polynomial time while guaranteeing logarithmic convergence with arbitrarily high probability:

Theorem 6.6. *Assume the agent's true reward is bounded by $R_{max} - \delta$ in every state. Let B_K^t denote the IRL space of reward functions implied by the constraints in K before the t-th iteration of Algorithm 6.1*, and let $b = \frac{1}{1-k}$ for some k such that $0 < k < \frac{1}{e}$. For any elicitation strategy that picks the average of $O(|S|)$ points sampled uniformly from B_K^t for $\widehat{R}$, with arbitrarily high probability, Algorithm 6.1* terminates with a solution to the policy teaching problem with unknown rewards or returns FAILURE if no solution exists after at most $1 + \lceil \log_b \lceil (\frac{R_{max}}{\delta})^{|S|} \rceil \rceil$ iterations.*

Theorem 6.7. *Each iteration of Algorithm 6.1* with the elicitation strategy from Theorem 6.6 is solvable in time polynomial in the number of states and actions.*

Sampling $O(|S|)$ points uniformly takes $O(|S|^4)$ steps of a random walk that requires $O(|S|^2)$ operations per step, so computing $\widehat{R}$ this way is $O(|S|^6)$ [6]. One can then find $\widehat{\Delta}$ satisfying $(\widehat{R} + \widehat{\Delta}) \in IRL_\epsilon^{\pi'}$ by solving a simple linear program.

6.3.4 Summary

We study the problem of policy teaching, in which the goal is to elicit a desired policy from an agent by providing rewards from a constrained space. Given unknown agent rewards, we constructed an algorithm that applies the active, indirect elicitation framework to quickly narrow down the space of possible rewards consistent with observed behavior. A centroid-based elicitation strategy guarantees convergence to a solution after few interactions, and is made tractable by applying appropriate sampling techniques.

Our analysis on policy teaching can be extended in a number of ways. Zhang et al. [108] considered a heuristic elicitation strategy based on maximizing the slack in IRL constraints. This approach does not provide logarithmic convergence guarantees, but is simpler (the elicitation strategy only requires solving a linear program), and achieved good empirical performance in simulation. Zhang et al. [108] also extended the elicitation algorithm to handle situations in which the interested party only observes the agent's actions instead of his policy, and in which the interested party only wishes to influence the agent's policy in a subset of the states.

Zhang and Parkes [107] considered the problem of *value-based policy teaching*, in which the goal is to provide limited rewards to elicit a policy that maximizes the interested party's value with respect to the *unknown* agent rewards. With this objective, computing the optimal incentives becomes NP-hard. The IRL space is no longer convex; while a similar active, indirect elicitation algorithm ensures convergence, logarithmic convergence cannot be guaranteed. Nevertheless, Zhang and Parkes [107] proposed a mixed-integer program for solving modest-sized instances, and presented simulation results showing that slack-based elicitation heuristics were still effective and elicited the best possible policy after few interactions.

6.4 Discussion

Our solution to the policy teaching problem demonstrates how designing experiments by reasoning about participants based on current models, and learning about participants based on observed behaviors, can form an automated, iterative design process that effectively solves automated environment design problems. Using the active, indirect elicitation framework, the elicitation strategy sets up a hypothesis about an agent's model parameters, and designs an experiment that either produces a desired outcome (e.g., the agent follows the desired policy) or rejects the hypothesis. If the hypothesis is rejected, the inference procedure refines the knowledge of model parameters, to eliminate not only the particular parameter values being tested but any model parameters that are inconsistent with observed behavior. Zhang et al. [103] showed how to generalize Algorithm 6.1 and the centroid-based elicitation strategy for other automated environment design problems, and extended the theoretical results about logarithmic convergence to any setting with observable decisions for which the space of model parameters considered during the elicitation process is convex.

While we assumed in the policy teaching setting that the agent's decision or policy is directly observable through his actions, in practice we may only have access to samples of agent actions. This implies, for example, that we cannot always set up a hypothesis that directly proves that a particular set of parameter values is not the agent's actual model parameters. In general, active, indirect elicitation algorithms may need to adopt a more probabilistic framework, where observed actions and outcomes are used to update beliefs over the underlying model parameters, but may never completely eliminate certain model parameters from consideration. As an example, Chapter 8 provides an active, indirect elicitation framework for automatically synthesizing crowdsourcing workflows, that adopts probabilistic beliefs.

Implicit in the active, indirect elicitation framework is the assumption that observations of behavior can be used to infer the agent's model parameters, and thus allow designers to better understand how participants make decisions based on which to more effectively design using learned models. In practice, models may be inaccurate and imprecise. The environment may be dynamic and involve changing factors that are outside of a designer's control but that nevertheless affect participant behavior. Some of these issues are explored in the next chapter, in which we consider an application of the active, indirect elicitation framework for automatically designing human computation tasks.

The active, indirect elicitation framework extends to settings with multiple participants, for which information about how participants may interact or affect one another's decisions can also be captured by an agent model and can likewise be refined by learning from observed behavior in response to well-chosen experiments. But with multiple participants there are new challenges, particularly in modeling the interaction among participants and how participants' individual actions can lead to complex outcomes. For example, a designer may need to reason about how the varied interests and abilities of participants can enable effective collaborative problem solving, or reason about how network effects may affect the adoption of a new feature. Considering multiple agents also brings into focus a broader range of elicitation processes, which includes the ability to select particular groups of users on which to conduct an experiment.

While we assume that participants in social and economic systems on the Web are myopically rational with respect to environment changes, participants can be forward looking and take actions that aim to induce the designer to select more desirable environment changes. For example, such situations have been observed in traditional labor markets, in which paid for performance workers purposely reduced their output to prevent the employer from using output measures to infer their actual ability and increase quotas or reduce pay.[10] When this occurs, the interested party cannot make inferences based on observed actions under the assumption that agents are acting straightforwardly, because the revealed information may not truthfully represent the agent's model parameters.

In certain settings, the interested party may be able to avoid such issues by committing to a goal (e.g., eliciting behaviors leading to a goal value that is above a set threshold) and by only exploring environment changes that benefit both the agent and the interested party. The interested party may undertake an active, indirect elicitation process, and either discover an environment change under which the agent behaves as desired, or if not then give up and reset to the base environment. If the agent prefers a potential environment change over the base environment, he may nevertheless reveal sufficient information through actions to ensure that a change

[10] This is often referred to as the *ratchet effect* in economics, and occurs when an employer cannot commit to *not* using revealed information to exploit a worker over time.

benefitting both parties is made.

In general, handling such issues requires reasoning carefully about the incentives of participants and the interested party. Whenever possible, an automated environment design procedure should aim to discover designs that create additional value and benefit both parties, and in the process mitigate concerns about non-straightforward behavior. As an example in which we try to achieve this goal, we consider in the next chapter the problem of automatically designing a human computation task, where we seek to identify task designs that lead to higher quality output at a fixed unit rate of pay.

Chapter 7

Automated Task Design

As discussed in earlier chapters, a central challenge in designing human computation systems is understanding how to construct decision environments that effectively attract participants and coordinate the problem-solving process. At a high level, the design of a human computation system consists of two components. One component is the design of incentives—social rewards, game points, and money—that helps to attract a crowd and encourage high quality work. The other component is the organization of individuals—the selection of participants, assignment of tasks, and design of interfaces and workflows—that helps to usefully harness individual efforts to advance a system's purpose. From the designer's perspective, the goal is to maximize the rate and quality of output, while minimizing the amount of human effort required and the cost incurred.

In this chapter, we apply ideas from automated environment design to tackle a common computational environment design problem that requesters face on Amazon Mechanical Turk: how should a task be designed so as to induce good output from workers? This question exemplifies both the incentive and organizational aspects of the design challenge. In posting a task, a requester decides how to break down the task into unit tasks (called HITs, for human intelligence tasks), how much to pay for each HIT, and how many workers to assign to each HIT. These design decisions shape the task environment, which may affect the rate at which workers view and complete unit tasks, as well as the quality of the resulting work.

There are a number of challenges involved in effectively designing a task for posting on Mechanical Turk. As we saw in the nutrition analysis example in Chapter 2, a notable problem is that the effect of design on the rate and quality of work is often imprecisely known *a priori*. Any design's effectiveness is likely dependent on the specifics of the task, and also the quality metric specified. While a designer may have some prior knowledge and be able to experiment with different designs, the design space is exponential in the number of design parameters while the number of experiments that can be performed is relatively small. Furthermore, Mechanical Turk is an inherently noisy and dynamic system, so any measurements obtained are affected in part by system conditions. Moreover, some statistics of interest, such as the number of active workers currently looking for tasks to perform, are unobservable by the requester.

Leveraging the active, indirect elicitation framework of automated environment design, we introduce a general approach for *automated task design*. In this approach, we construct models for predicting the rate and quality of work. These models are trained on worker outputs over a set of designs, and are then used to optimize a task's design. We demonstrate our approach on an image labeling task, for which we aim to maximize the number of quality labels received, subject to budget constraints. We consider two measures of quality: one based on the number of distinct labels received, and another based on the number of distinct labels received that match an external gold standard.

In our experiments, we find that simple models can accurately predict the output per unit task for both quality metrics, and that the models generate different designs depending on the quality metric we care about. For predicting the rate of work, we observe that a task's completion time is correlated with the amount of work requested per dollar paid, and depends on the time of day when a task is posted. But despite these effects, we find that due to varying system conditions on Mechanical Turk, the task completion time is nevertheless difficult to predict accurately and can vary significantly even for the same design. Focusing on using the quality prediction models for design, we find that for the same budget and rate of pay, optimized designs generated by our models obtain significantly more quality tags on average than baseline designs for both quality metrics.

Section 7.1 reviews related work. Section 7.2 describes the Mechanical Turk marketplace and introduces a general approach for automated task design. Section 7.3 describes the image labeling task. Before exploring different designs for this task, Section 7.4 details an experiment to capture the amount of variability on Mechanical Turk, where we post the same task design multiple times under varying system conditions. Section 7.5 discusses our initial experiments and reports on the performance of models for predicting the rate and quality of work. We consider optimizing the task design based on trained models in Section 7.6, and compare the performance of optimized designs to baseline designs that pay at the same rate. Section 7.7 discusses the implications of our experiments for automated task design and outlines the possibilities and challenges moving forward.

7.1 Related Work

Studies on the effect of monetary incentives on worker performance found that monetary incentives attracted Mechanical Turk workers (Turkers) to perform more HITs of a task [32, 66, 78, 11] but did not affect the quality of work [66, 78]. In our image labeling task, we also find that tasks are completed more quickly at higher rates of pay. While we find that we can accurately predict the quality of work without factoring in compensation, we do not study the effect of pay on work quality and focus instead on finding effective designs that elicit good output at a fixed rate of pay.

A number of studies have also considered the effect of non-monetary interventions on work quality. Dow et al. [23] showed that asking Turkers to self-assess their work against key performance criteria can improve work quality. Shaw et al. [85] showed that when coupled with monetary incentives, asking Turkers to think about their peers' responses can also improve work quality. Findings on the effect of intrinsic motivation on work quality are mixed; whereas Chandler and Kapelner [12] found that framing a task as being for a good cause did not induce Turkers to produce higher quality solutions, Rogstadius et al. [78] found in their experiments that doing so significantly improved solution quality.

Other studies have considered designing Turk tasks by organizing workers and aggregating output. Snow et al. [87] considered a number of different natural language annotation tasks, and showed that annotations based on the majority output among a group of Turkers is comparable in quality to expert annotations, but is cheaper and faster to obtain. Su et al. [90] considered the effect of qualification tests on worker output and showed that workers with higher test scores achieve higher accuracy on the actual task. In an orthogonal direction, this chapter focuses on effectively distributing work across identical, parallel subtasks.

Human-powered database systems that recruit a crowd to perform operations such as filters, sorts, and joins are often concerned with efficiency and interested in optimizations that make better use of human effort. Marcus et al. [63, 62] introduced a declarative workflow engine called Qurk and proposed optimizations such as batching tasks and pre-filtering tables before joins. Parameswaran et al. [70] introduced a crowdsourced database system called Deco, and demonstrated that the choice of query execution plan can significantly affect performance. In these systems, having automated procedures that can learn and reason about the crowd's performance on tasks can potentially provide a means for *query optimization*, that seeks to identify efficient, crowd-tailored query plans.

Several works have applied decision-theoretic planning techniques to control the request for additional work in human computation systems. Kamar et al. [43] demonstrated how predictive models can be used to control the request of additional votes for classifying celestial objects in Galaxy Zoo. Dai et al. [16, 17] introduced TurKontrol, a system for controlling the request of additional voting or improvement tasks based on costs and the inferred work quality. In this chapter, we focus on a complementary challenge of learning about workers to best design individual tasks.

7.2 Automated Task Design on Mechanical Turk

7.2.1 Mechanical Turk

We first review the design environment presented by Amazon Mechanical Turk (`www.mturk.com`). Mechanical Turk is a crowdsourcing marketplace for work that requires human intelligence. Since its launch in 2005, a wide variety of tasks have been posted and completed on Mechanical Turk. Example tasks include audio transcription, article summarization, and product categorization. Increasingly, Mechanical Turk is also attracting social scientists who are interested in performing laboratory-style experiments [33].

On Mechanical Turk, a *requester* posts jobs for hire that registered workers can complete for pay. A *job* is posted in the form of a group of *HITs* where each HIT represents an individual unit of work that a worker can accept. A requester can seek multiple *assignments* of the same HIT, where each assignment corresponds to a request for a unique worker to perform the HIT. The requester sets the lifetime during which the HITs will be available and the amount of time a worker has to complete a single HIT. The requester can also impose a qualification requirement for a worker to be eligible to perform the task.

When choosing a task to perform, a worker is presented with a sorted list of available jobs, where for each job the title, reward, expiration time, and number of HITs available are displayed. The list can be sorted by the number of HITs available (the default), the reward, creation time, or expiration time. Workers can see a brief task description by clicking the title, or choose to “view a HIT in this group” to see a preview of a HIT. At this point the worker can choose to accept or skip the HIT. If the HIT is accepted, it is assigned to that worker until it expires or is submitted or abandoned. Workers are not provided with additional information on the difficulty of tasks by the system, although there is evidence of workers sharing information on tasks and requester reputation via browser extensions and on Turk-related forums.[1]

Upon receiving completed assignments, the requester determines whether to approve or reject the work. If an assignment is rejected, the requester is not obligated to pay the worker. While tasks vary greatly in pay and the amount of work required, the *reward* per HIT is often between \$0.01 to \$0.10, and most individual HITs require between a few seconds to a few minutes to complete. There are thousands of job requests posted at any given time, which correspond to tens and hundreds of thousands of available HITs. For each HIT completed, Amazon charges the requester 10% of the reward amount or half a cent, whichever is more.

7.2.2 An Automated Approach to Task Design

An exciting aspect of Mechanical Turk as a human computation platform is that it allows a requester to post arbitrary tasks for a large population of workers to complete. A requester has the freedom to design his or her task as desired, with the aim of eliciting good effort from workers toward generating useful work. The task design allows a requester to optimize tradeoffs among the rate of work, the quality of work, and the cost of work. While some of the qualitative aspects of tradeoffs are well understood (e.g., paying more will increase the rate of work, both because more

[1] See `http://turkopticon.differenceengines.com/` and `http://www.turkernation.com/`, respectively.

workers will want to accept HITs and that each worker will want to complete more HITs [32]), optimizing the design to achieve particular tradeoffs requires a quantitative understanding of the effect. The effect of non-monetary aspects of task design (e.g., the division of a task into HITs and assignments) on the quality and quantity of work is less well understood, even qualitatively. Such effects are likely to be specific to the task at hand, and depend on a particular requester's goals and constraints.

We advance an automated approach to task design based on the active, indirect elicitation framework of automated environment design. For a given task, we first experiment with different designs and use the workers' output and measurements of system conditions to learn a *task-specific model* of the effect of design on the rate and quality of work.[2] We then use learned models to optimize for good designs based on their predictions. From the automated environment design perspective, we are interested in whether a model learned from observing worker performance can effectively guide the search for better designs.

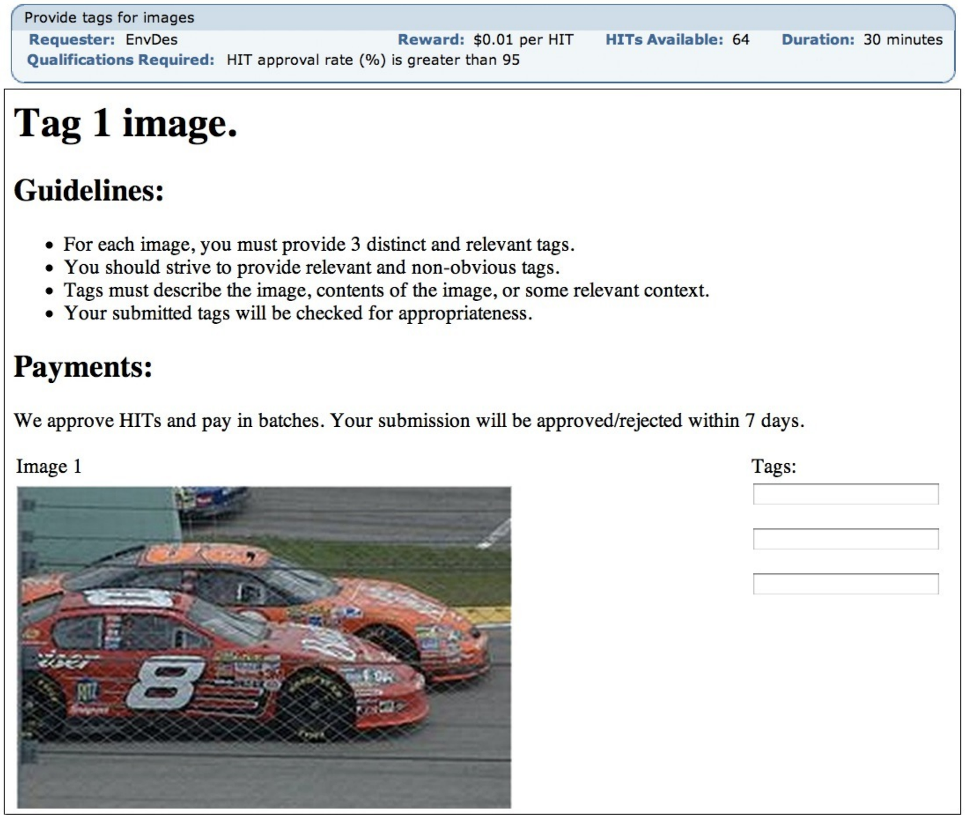
In the rest of the chapter, we consider as a case study the problem of automatically designing an image labeling task. We describe the task and its design space in the next section, and then apply the following steps to discover an effective design:

1. Estimate variances in target metrics with a baseline design (Section 7.4)

2. Explore the design space with experiments (Section 7.5)

3. Fit models to the experimental data (Sections 7.5.1 and 7.5.2)

[2] In the general active, indirect elicitation framework, learned information can be incorporated after each experiment and can inform which experiments to conduct thereafter. For simplicity, the elicitation strategy we consider in this setting simply picks a set of experiments to run in *batch*. The inference procedure then updates the model after all experiments are completed.



Provide tags for images

Requester: EnvDes Reward: $0.01 per HIT HITs Available: 64 Duration: 30 minutes

Qualifications Required: HIT approval rate (%) is greater than 95

Tag 1 image.

Guidelines:

- For each image, you must provide 3 distinct and relevant tags.
- You should strive to provide relevant and non-obvious tags.
- Tags must describe the image, contents of the image, or some relevant context.
- Your submitted tags will be checked for appropriateness.

Payments:

We approve HITs and pay in batches. Your submission will be approved/rejected within 7 days.

Image 1

Tags:

Figure 7.1: A HIT of the image labeling task

4. Optimize the target metrics given the fitted models (Section 7.6.1)

5. Run experiments using the optimized task parameters to validate our approach (Section 7.6.2)

7.3 The Image Labeling Task

We consider an image labeling task in which workers are asked to provide relevant labels (or equivalently, tags) for a set of images. Each HIT contains a number of images, and for each image, requests a particular number of labels for that image.

Workers are informed of the number of images and number of labels required per image within the guidelines provided in the HIT, and are asked to provide "relevant and non-obvious tags." Workers can provide tags containing multiple words, but this is not required nor specified in the instructions. See Figure 7.1 for a sample HIT that requests three labels for one image. Example labels for this image include "NASCAR," "race cars," "red," "Dale Earnhardt Jr.," "eight," and "tires."

We obtained a large dataset of images from the ESP game,[3] which contains 100,000 images and labels collected through gameplay. From this dataset, we use images that contain at least ten labels, of which there are 57,745. Of these, we have used 11,461 images in our experiments. Any particular image we use appears in only one HIT.

We consider two metrics for judging the quality of labels received from workers. One metric counts the number of unique labels received, and is thus concerned with the number of labels collected. The other metric counts the number of labels received that also appear as labels in our gold standard (GS) from the ESP dataset. Since the gold standard labels are those most agreed upon in the ESP game, they are labels that are likely to capture the most noticeable features of an image.

To compute these metrics, we first preprocess labels to split any multi-word labels into multiple single-word labels and convert upper case letters to lower case. We then apply the standard Porter Stemming Algorithm [75] to normalize worker and gold standard labels. This ensures that labels such as "dog" and "dogs" are considered the same label, which is useful for our measure of uniqueness and for comparing received labels to the gold standard. Finally, we remove *stop words* such as "a" and

[3] `http://www.cs.cmu.edu/~biglou/resources/`

"the," which account for 0.9% of gold standard labels and 4.6% of distinct labels collected.[4]

In designing the image labeling task, a designer can decide on the reward per HIT, the number of images and tags requested per image per HIT, the total number of HITs, the number of assignments per HIT, the time allotted per HIT, and the qualification requirements. The requester's goal is to maximize the number of useful labels received as judged by the quality metric of interest, subject to any time and budget constraints. For example, a requester may have $5 to spend, and aims to collect as many unique tags as possible within the next six hours. One can compare two different designs based on the amount of useful work completed within a certain time frame, or by examining the tradeoff between the work completed per dollar spent and the rate of work.

While each design variable may have an effect on output, we focus our efforts on designing the reward per HIT, the number of images per HIT, the number of labels requested per image, and the total number of HITs. For our experiments, we fix the time allotted per HIT at 30 minutes (the default), but do not expect workers to spend more than a few minutes per HIT. We fix the number of assignments per HIT at 5; this gives us multiple sets of labels per image and will enable a study of the marginal effects of recruiting an additional worker to a HIT on the quality of output in future research. We require all workers to have an approval rate of at least 95%, such that only workers with 95% or more of their previously completed HITs approved are allowed to work on our task.

[4] We used a short, conservative list of stop words from `http://www.textfixer.com/resources/`.

When posting tasks, we collect measurements of worker views and accepts over time, the amount of time a worker spends on a HIT, and the value of output as judged by our quality metrics. We also collect information on system conditions such as the time of day, the number of HITs available on Turk, the page position of our posting in different list orderings, and the number of completed HITs overall in Mechanical Turk. The last statistic is not available directly, and is estimated by tracking the change in the number of HITs available for tasks in the system at two minute intervals.

7.4 Measuring Output Variability

Before considering the effect of design on output, we first report on the amount of variability in the output from Mechanical Turk when using *a fixed task design*. This lets us know how much variance to expect from the system, and allows us to study the effect of system conditions on output.

By observing and following common practice on Mechanical Turk, we selected a design for which each HIT has a reward of \$0.01, contains one image, and requests three labels. We posted a group of 20 HITs at a time, and posted 24 groups of the same task design from 4/12/2010 to 4/20/2010. Each group of HITs was allowed to run for approximately eight hours, and groups of HITs were posted sequentially around the clock. All groups had at least 75% of the assignments completed, with 18 of the 24 groups finishing before the time expired.

Table 7.1 summarizes the mean and standard deviation of the rate and quality of output along a number of measurements.[5] The task took 5 hours and 30 minutes

[5] We measure the completion time of an unfinished task as the time until the job expires (∼8

Statistic	Mean	Standard Deviation
Time to 50% completion (min)	129.54	95.13 / 73%
Time to 100% completion (min)	330.44	124.93 / 38%
Total # of unique tags	264.56	18.06 / 7%
Total # of unique tags in GS	98.56	9.50 / 10%
# of unique workers	13.33	2.99 / 22%
Time to complete a HIT (s)	74.79	25.12 / 34%

Table 7.1: Statistics on a group of image labeling tasks with 20 HITs that was posted 24 times between 4/12/2010 and 4/20/2010. Each HIT pays $0.01 and requests three labels for one image.

to complete on average, with the quickest run completing in just under 52 minutes and the longest run taking 8 hours and 37 minutes. Unlike task completion time, the number of unique labels received and the number of such labels that are in the gold standard vary much less, suggesting that the quality of output from workers remains relatively constant under different system conditions.

One possible explanation for the significant variation in completion time is that the activity level of workers on Mechanical Turk varies over time. While we do not know how many workers are active on Mechanical Turk at any given time, it is reasonable to think that activity level is correlated with time of day. That is, the system is likely more active during particular "work hours" than at other times. In Figure 7.2 we plot the relationship between the posting time and the time by which 50% or 100% of the tasks were completed. We observe that jobs posted between 6AM GMT and 3PM GMT were completed most quickly; this corresponds to posting between 2AM to 11AM EST in the United States and 11:30AM to 8:30PM IST in India, the two countries that provide 80% of workers on Mechanical Turk [39]. Given that these

hours), and only measure the number of tags and unique workers for completed tasks.

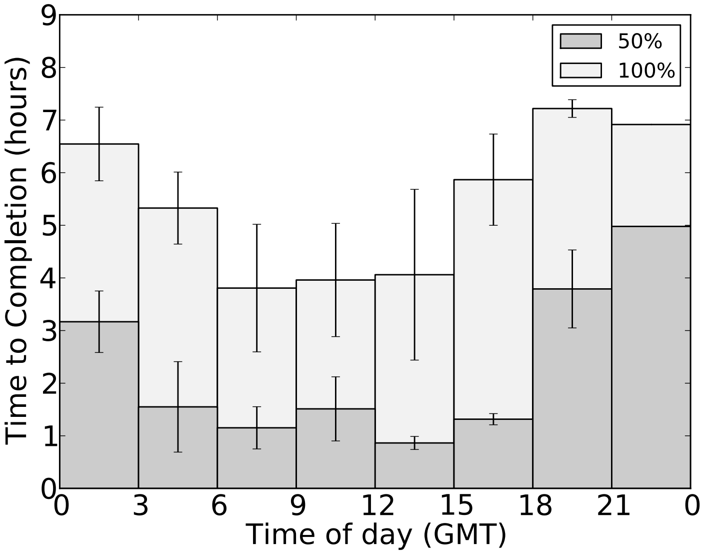


Figure 7.2: The effect of posting time on time until 50% and 100% completion. Bins depict the average completion time of runs posted within a three hour period and error bars represent the standard error. Experiment conducted from 4/12/2010 to 4/20/2010.

times correspond to waking hours in India, we expect most of the workers interested in this task to be from India. We geolocated workers based on their IP addresses by using the Linux shell command **whois**. Of the IP addresses for which we can determine the country of origin (247 out of 307), 62% were from India and 23% were from the US, which is consistent with our intuition.

7.5 Initial Experiments and Behavioral Models

From the variability measurements we learned that the completion time of a task may be highly variable, and may be difficult to predict accurately even for a fixed design. While some of the time variability can be explained by the time of day in which the task is posted, there is still a substantial amount of residual noise. In contrast, we find that the quality of work does not vary much with system conditions. Based on these observations, we expect that task design may have a large effect on the quality of work, but will only partially influence the rate of work.

In order to understand the effect of design on worker output, we developed models for predicting the quality of labels received per HIT and the completion time. We performed a series of 38 initial experiments—which serves as our training data—in which we varied the task's design (or configuration) by changing the reward (R), the number of images (N_{pic}) and number of labels per image per HIT (N_{tag}), and the number of HITs (N_{hits}). We considered rewards in the range of \$0.01 and \$0.10 per HIT, and varied the number of images and tags requested between 1 and 10. In choosing configurations, we aimed to cover a large range of values along each dimension, and to vary the total number of tags requested per dollar pay, i.e., $N_{pic}N_{tag}/R$. For the most part we considered jobs that consist of groups of 20 HITs (in 31 configurations), but also included a few jobs containing 30, 150, 500, and 1000 HITs, respectively. Configurations were randomly ordered and allowed to run until completion. They were automatically posted in series over a three week period from 2/2/2010 to 2/24/2010 with no gaps between postings. We fixed the number of assignments (N_{asst}) requested per HIT at five, and required all workers to have an approval rate of at least 95%.

In considering models for predicting the rate and quality of work, we measured the goodness of fit by reporting the coefficient of determination (R^2), the root mean square error (RMSE), the root relative square error (RRSE), and the mean absolute error (MAE), between predicted and actual output. All statistics are computed for the hold-out data via leave-one-out cross-validation.

7.5.1 Predicting Label Quality

We consider models for predicting the average number of quality labels received from workers. A summary of model coefficients and fitness is presented in Table 7.2.

Predicting Unique Tags

For predicting the average number of unique tags that are received per assignment (N_{unique}),[6] we hypothesized that we would experience diminishing marginal returns as we request more tags per image, suggesting the following model:[7]

$$N_{unique} = \beta N_{pic} log(N_{tag}) + \epsilon \tag{7.1}$$

We find that the model's predictions are somewhat accurate, with $R^2 = 0.77$. We also considered a model without diminishing marginal returns in the number of labels requested:

$$N_{unique} = \beta N_{pic} N_{tag} + \epsilon \tag{7.2}$$

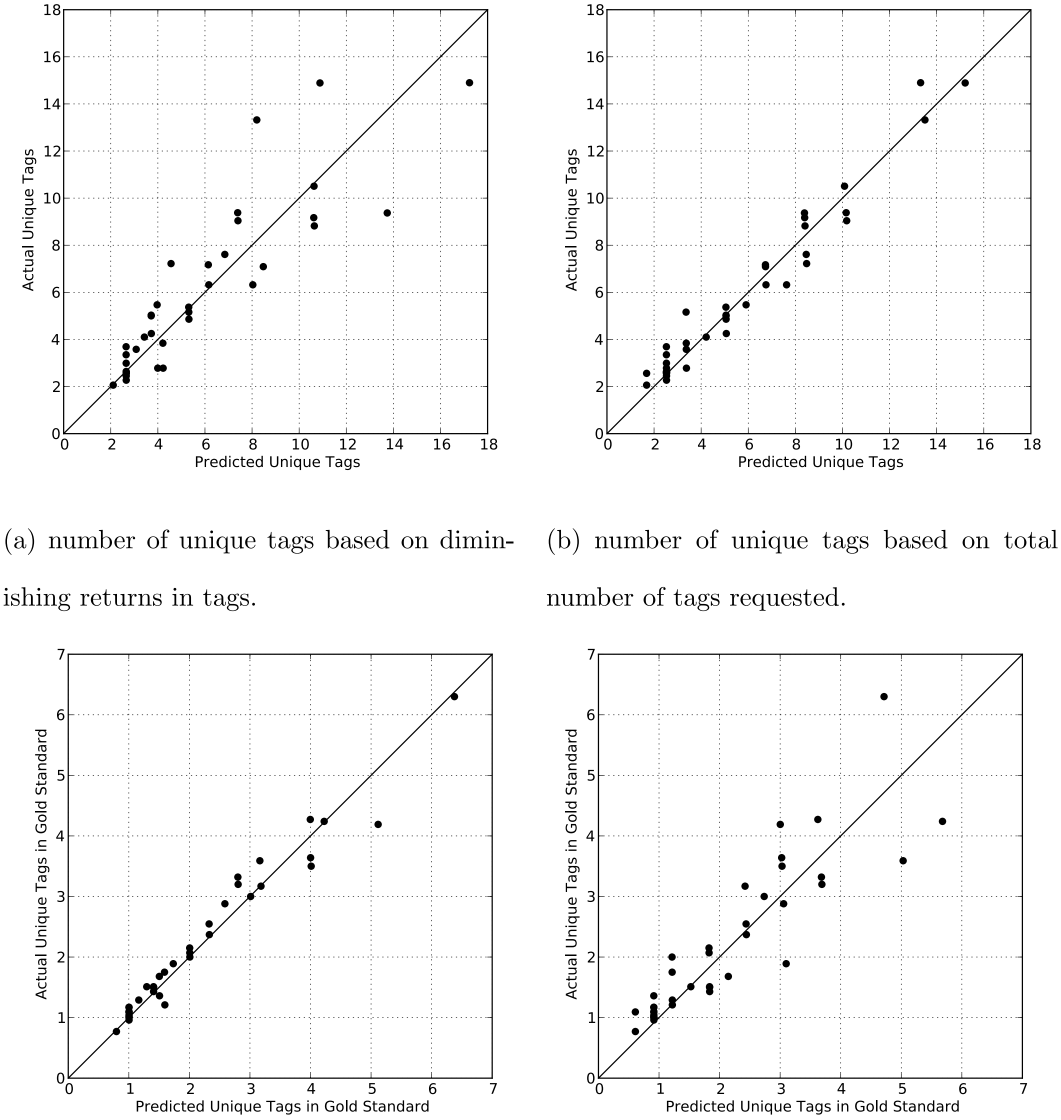
Surprisingly, we observe a significantly better fit, with $R^2 = 0.96$; see Figures 7.3(a) and 7.3(b) for a comparison between the two models' predictions. The model without diminishing returns suggests that the proportion of overlap in tags entered across the five assignments is invariant to the number of tags requested, and that at least within the range of values in our training data we do not observe workers running out of tags to describe an image.

[6] We compute the per assignment contribution by dividing the number of quality tags collected per HIT by the number of assignments, which is fixed at five.

[7] When taking a log, we smooth the input data by adding 1 to the number of tags (N_{tag}) to ensure the feature has weight instead of evaluating to zero.

	Estimated Model		R^2	RMSE	RRSE	MAE
	$N_{pic}N_{tag}$	$N_{pic}\log(N_{tag})$				
Diminishing returns in tags		1.9157 (0.0743)	0.7681	1.6617	0.4816	1.1341
Linear in tags	0.8426 (0.0140)		0.9576	0.7105	0.2059	0.5491
Diminishing returns in tags		0.7241 (0.0115)	0.9576	0.2574	0.2057	0.1743
Linear in tags	0.3050 (0.0115)		0.7652	0.6064	0.4853	0.4406

Table 7.2: Summary of model coefficients with standard errors (in parenthesis) and goodness of fit for predicting the average number of quality labels received per assignment. The top two models predict the number of unique tags collected, and the bottom two models predict the number of unique tags collected that are in the gold standard.



(a) number of unique tags based on diminishing returns in tags.

(b) number of unique tags based on total number of tags requested.

(c) number of unique tags in gold standard based on diminishing returns in tags.

(d) number of unique tags in gold standard based on total number of tags requested.

Figure 7.3: Predicted vs. actual number of quality tags received per assignment

Predicting Unique Tags that are in Gold Standard

For predicting the average number of unique tags received per assignment that are in the gold standard (N_{gs}), we again hypothesized that there would be an effect of diminishing marginal returns as we request more tags per image. Since there is a limited number of tags per image within the gold standard with which the collected tags can match, we would expect the effect of diminishing returns to be much stronger than for our other quality metric. We consider the following model:

$$N_{gs} = \beta N_{pic} \log(N_{tag}) + \epsilon \tag{7.3}$$

The prediction is highly accurate, with $R^2 = 0.96$. The model's fit is significantly better than the fit of a model without diminishing returns ($R^2 = 0.77$); see Figures 7.3(c) and 7.3(d).

7.5.2 Predicting Completion Time

Continuing, we consider models for predicting completion time based on a task's design. Table 7.3 provides a summary of model coefficients and fitness.

Intuitively, a task is more attractive if the pay is high but the amount of work is low. Given similar amounts of work, we would expect the number of tags requested per dollar pay (rate of pay) to be correlated with a task's completion time. We consider all 31 configurations with 20 HITs from our training data, and predict the 50% completion time ($T_{1/2}$) and 100% completion time (T) using the following model:

$$T = \beta_0 + \beta_1 \frac{N_{pic} N_{tag}}{R} + \epsilon \tag{7.4}$$

	Estimated Model				R^2	RMSE	RRSE	MAE
	$\frac{N_{pic}N_{tag}}{R}$	$\cos(t)$	$\sin(t)$	Intercept				
Pay	25.58 (6.81)			-373.45 (3452)	0.45	12418	0.88	7898
Pay + Posting Time	21.47 (5.26)	12173 (2542)	1020 (2265)	5177 (2865)	0.70	9819	0.70	7549
Pay	49.79 (8.06)			4955.10 (4082)	0.68	14914	0.72	11583
Pay + Posting Time	44.71 (6.35)	13775 (3072)	-3104 (2738)	10801 (3463)	0.79	12630	0.61	9655

Table 7.3: Summary of model coefficients with standard errors (in parenthesis) and goodness of fit for predicting completion time (in seconds). The top two models predict the 50% completion time, and the bottom two models predict the 100% completion time.

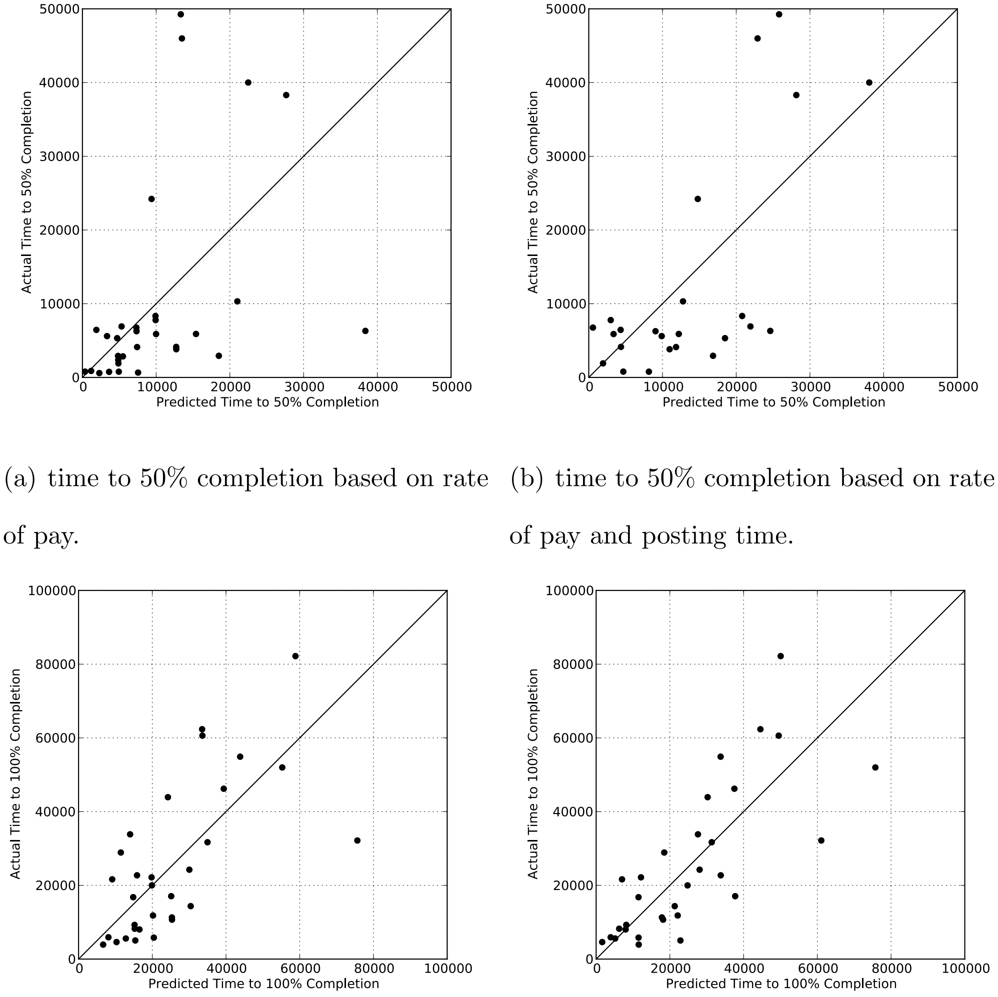
We see that the rate of pay is correlated with the completion time, with $R^2 = 0.68$ for predicting 100% completion. The correlation is weaker for predicting 50% completion time, with $R^2 = 0.45$.

From the results of our variability study, we also expect the time of posting to affect the completion time. As we saw in Figure 7.2, the effect of time of day on completion time is sinusoidal. To incorporate this effect into our model, we convert the time of day to an angle t between 0 and 2π, corresponding to 0:00 GMT and 24:00 GMT respectively, and then encode it as two units, $\cos(t)$ and $\sin(t)$. This encoding scheme ensures that each time of day has a distinct representation and that the values for times around midnight are adjacent. Adding these time variables, we fit the following model:

$$T = \beta_0 + \beta_1 \frac{N_{pic} N_{tag}}{R} + \beta_2 \cos(t) + \beta_3 \sin(t) + \epsilon \tag{7.5}$$

We observe an improvement in the fit, with $R^2 = 0.79$ for 100% completion time, and $R^2 = 0.70$ for 50% completion time; see Figure 7.4 for a comparison between the models' predictions. This improvement is more significant for predicting 50% completion time (R^2 from 0.45 to 0.70) than for 100% completion time (R^2 from 0.68 to 0.79). One possible explanation is that the effect of the posting time diminishes when HITs are posted for a longer time frame that includes other times of the day.

The fit of these models suggests that the rate of pay and the time of posting are correlated with the completion time, but that there is still a substantial amount of unexplained variance. To use these models for prediction and design, it would be useful to consider not only the expected completion time, but also to be mindful of the variance in the prediction. Furthermore, the current models are only trained on



(a) time to 50% completion based on rate of pay.

(b) time to 50% completion based on rate of pay and posting time.

(c) time to 100% completion based on rate of pay.

(d) time to 100% completion based on rate of pay and posting time.

Figure 7.4: Predicted vs. actual time until 50% and 100% completion (in seconds).

configurations with 20 HITs, and do not incorporate the effect of varying the number of HITs. We leave the exploration of these directions for future work, and for now focus on using the quality prediction models for design.

7.6 Design Experiment

The initial experiments provide us with an understanding of how workers respond to different designs and thus serve as the building blocks for effective task design. Even at the same level of desirability to workers—e.g., as measured by the pay per tag, or more generally, the estimated pay per hour—we expect some designs to induce more quality output than other designs. We now investigate whether the learned models can help us make informed design decisions for particular quality metrics of interest.

7.6.1 Design Optimization and Experiment Setup

We consider a simple design experiment in which we compare different designs at a fixed pay per tag. We focus our comparison on the number of quality labels received (per dollar spent), and do not concern ourselves with the rate at which work completes.[8] Fixing the rate of pay allows us to compare designs based on the kind of work they request, and removes the effect of assigning more work at a lower rate of pay to get more quality labels from confounding the comparison.

We consider experiments at two pay rates: a low rate that pays 1¢ for every three tags, and a high rate that pays 1¢ per tag. For each pay rate, we compare the

[8] In practice, we can set the rate of pay based on how quickly we want work to get done. But since time is not considered in this experiment, fixing the rate of pay allows for a fair comparison between designs.

output of *baseline designs* to designs optimized for each of our two quality metrics. Baseline designs are chosen by observing common practice in image labeling tasks on Mechanical Turk, which typically requests three or four tags for a single image within each HIT. Each design is given a budget of \$5, which must account for fees paid to Amazon as well as payments to workers. As in our initial experiments, the number of assignments per HIT (N_{asst}) is fixed at 5.

To optimize the task design, we choose values for the reward per HIT (R), number of images per HIT (N_{pic}), number of tags requested per image (N_{tag}), and the total number of HITs (N_{hits}), in order to maximize the total number of quality tags received as predicted by the model with the best fit, subject to budget and rate of pay constraints. We consider rewards in the range of \$0.01 to \$0.10 per HIT, and the number of images and tags requested per image in the range of 1 to 10. For example, the following formulation captures the optimization problem for finding a design that maximizes the total number of unique tags received as predicted by our model, subject to a \$5 budget and a pay rate of \$0.01 per tag:

$$\max_{R,N_{pic},N_{tag},N_{hits}} 0.8426 N_{pic} N_{tag} N_{hits} N_{asst} \tag{7.6}$$

$$N_{HIT} N_{asst}(R + \max(0.1R, 0.005)) \leq 5 \tag{7.7}$$

$$R/N_{pic} N_{tag} = 0.01 \tag{7.8}$$

Constraint 7.7 ensures that the cost of the design stays within budget, and constraint 7.8 ensures that the pay per tag is \$0.01. The max term in the budget constraint corresponds to Mechanical Turk's per assignment fees, which is 10% of the reward or half a cent, whichever is more.

Table 7.4 summarizes the baseline and optimized designs for both pay rates and

	pay/tag	R	N_{pic}	N_{tag}	N_{hits}	N_{asst}	Posting Fees	Total Cost
low pay baseline	$\frac{1}{3}$¢	\$0.01	1	3	66	5	\$1.65	\$4.95
optimized for N_{unique} (low pay)	$\frac{1}{3}$¢	\$0.06	9	2	15	5	\$0.45	\$4.95
optimized for N_{GS} (low pay)	$\frac{1}{3}$¢	\$0.03	9	1	28	5	\$0.70	\$4.90
high pay baseline	1¢	\$0.04	1	4	22	5	\$0.55	\$4.95
optimized for N_{unique}, N_{GS} (high pay)	1¢	\$0.10	10	1	9	5	\$0.45	\$4.95

Table 7.4: Baseline and optimized designs for an image labeling tasks with a $5 budget.

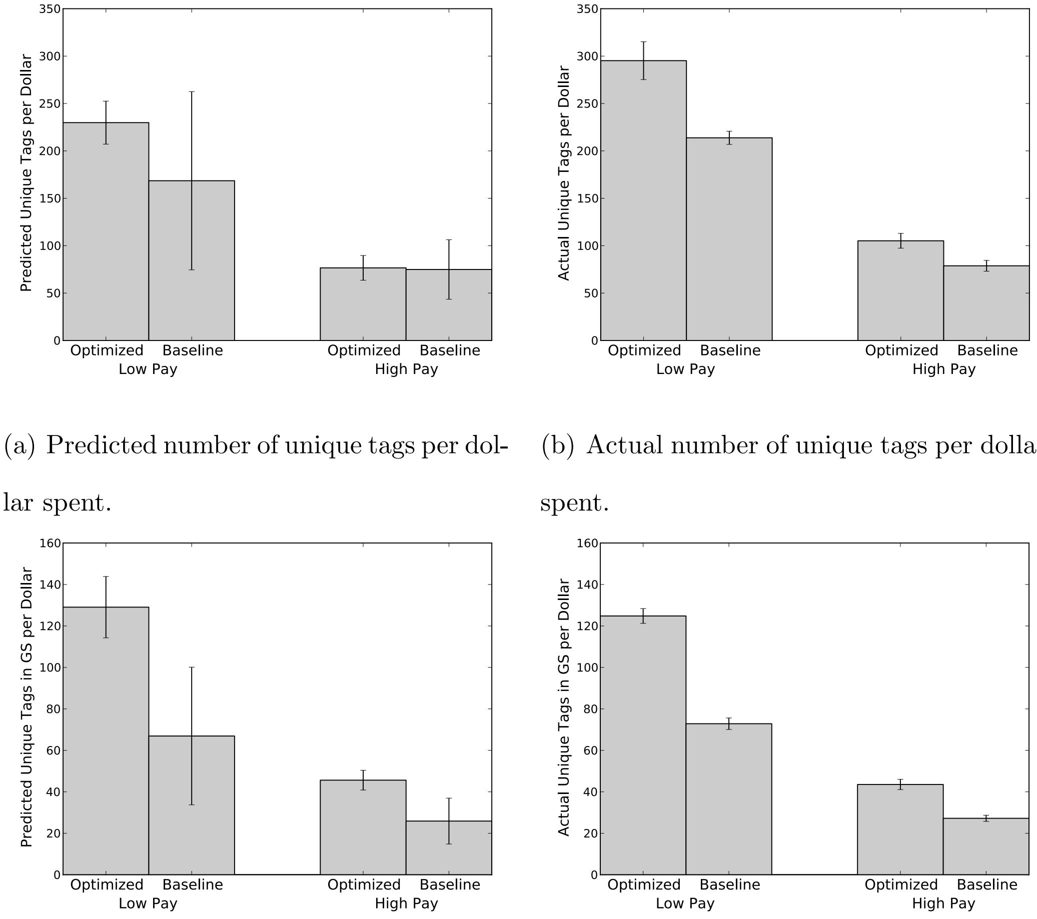
quality metrics. For the low pay rate, we consider a baseline design that requests 3 tags for one image, which is the same design that we had adopted for measuring variability (but with more HITs). For maximizing the number of unique tags collected, we see that the optimized design attempts to save on posting fees by putting more work into a HIT and paying more per HIT, which allows for more tags to be requested. For maximizing the number of unique tags that are in the gold standard, the optimized design avoids diminishing returns by requesting 1 tag per image, and also saves on posting fees by putting more work in a single HIT.

For the high pay rate, we consider a baseline design that requests 4 tags for one image. Here the optimized designs for the two quality metrics are the same. More work is put into each HIT to save on posting fees (hitting the upper bound on reward per HIT) and only 1 tag is requested per image to avoid diminishing returns.

Figures 7.5(a) and 7.5(c) show the models' predictions with bars representing the 95% prediction interval for these designs. We see that the difference in the predicted numbers of unique tags per dollar spent between baseline and optimized designs is small, since the benefits of the optimized design comes only from savings in posting fees. By avoiding diminishing returns in tags, designs optimized for the numbers of unique tags that are in the gold standard are expected to perform significantly better.

We post five groups of each baseline and optimized design in round-robin order. Each group ran initially for 6 hours and was allowed to finish at a later time if needed.[9]

[9] We initially posted the baseline designs between 3/25/2010 and 3/29/2010, and the optimized designs between 4/22/2010 and 4/26/2010. While almost all trials of the high pay configurations completed within this time frame, many of the low pay configurations did not; these configurations were ran to completion between 4/29/2010 and 5/7/2010.



(a) Predicted number of unique tags per dollar spent.

(b) Actual number of unique tags per dollar spent.

(c) Predicted number of unique tags in gold standard per dollar spent.

(d) Actual number of unique tags in gold standard per dollar spent.

Figure 7.5: Predicted and actual number of quality tags received per dollar spent for baseline and optimized designs. Error bars in predictions indicate the 95% prediction intervals, and error bars in results represent the standard error over five runs of each design.

7.6.2 Results

Figures 7.5(b) and 7.5(d) show the average number of unique tags and the average number of unique tags in the gold standard received per dollar spent, with bars capturing the standard error of the mean.

In all comparisons, we find that the optimized designs received more quality tags than baseline designs. The optimized designs for unique tags received 38% more tags in the low pay condition, and 33% more in the high pay condition. For collecting unique tags that are in the gold standard, the optimized designs received significantly more quality tags than the baseline comparisons, with 71% more in the low pay condition and 60% more in the high pay condition. For all baseline and optimized designs, the actual number of gold standard tags received is very close to our model's predictions (within 11%), and well within the prediction intervals.

Interestingly, our optimized designs received significantly more unique tags than our models predicted: 28% more in the low pay condition and 38% more in the high pay condition. One possible explanation is that our model underpredicts the number of unique tags when the number of tags requested per image is low, as is the case in our designs. After checking the model's predictions on the training data, we noticed that our model underpredicts for 10 out of the 11 configurations that request one or two tags per image (by 15% on average). Our model also underpredicted the number of unique tags obtained by the baseline in the low pay condition by 27%, suggesting that the model may need to be refined to improve prediction accuracy. Nevertheless, the information contained in the model was still helpful in discovering optimized designs that significantly outperform the baseline designs.

7.7 Discussion

By collecting data about how workers respond to designs in our initial experiments, we are able to construct models that can accurately predict worker output in response to different designs. These models can then be used to optimize a task's design, subject to designer constraints such as budget and rate of pay, to induce quality output from workers. The results from our experiments show that designs that are optimized based on learned models obtain significantly more high quality labels than baseline comparisons.

There are a number of possible extensions to this work. We would like to understand the effect of distributing work across multiple assignments on the quality of output, and to include the number of assignments as a design variable. We are also interested in revisiting models for predicting the rate of work, and incorporating them to design with respect to time-related tradeoffs. One possible direction is to learn the relative rates at which work completes for different designs, which may be sufficient for accurately predicting the relative output between designs. Furthermore, while we focus here on the design of a task with identical, parallel subtasks, we are interested in developing a general approach for automating the design of human computation algorithms and workflows. We discuss this in the next chapter.

We believe the active, indirect elicitation approach of learning from observations of behavior to optimize designs can be effectively used to design a variety of tasks, with respect to different performance metrics, and in richer design spaces. While linear regressions were used for this work, other modeling approaches and methods from machine learning and statistics can be incorporated into the design process. The models of behavior need to be specific to the particular task and performance metric at hand. Constructing accurate models will likely require drawing from an understanding of the task domain and the population of workers, and learning from experimentation.

In addition to accurate models, we need methods that help to discover effective designs quickly after only a few experiments. While we trained our models on a set of manually picked designs and then used these models to optimize the design, we can develop elicitation strategies that automatically pick subsequent experiments in a way that drives the search for better designs. In the next chapter, we develop a general method for automatically synthesizing workflows in which the system optimizes the choice of experiments to maximize the value of information obtained.

Chapter 8

Automated Workflow Synthesis

In the last chapter, we introduced an approach for automating the design of human computation tasks with identical, parallel subtasks. In this chapter, we develop a general framework for automating the synthesis of human computation algorithms and workflows involving heterogeneous tasks.

There are often many ways to coordinate a crowd to solve a problem. Different human computation algorithms or workflows embody different approaches, and may utilize distinct tasks or allocate effort differently among the same tasks. Given a space of possible human computation algorithms for solving a problem, figuring out which algorithms are the most efficient requires understanding how the crowd performs on individual tasks within an algorithm and how this in turn influences the quality of the final solution and the cost of effort incurred. The goal of the designer is to discover efficient workflows that make effective use of human effort to achieve high quality solutions, and in doing so take into account crowd characteristics and any time or resource constraints the designer may face.

Since inputs from the crowd are inherently noisy and submitted answers can be incorrect, the output from any task is *probabilistic*. An algorithm may apply quality control mechanisms that use redundancy or voting to mitigate potential errors in tasks, which helps to mitigate errors in the final solution but incurs additional cost of effort. Choosing an algorithm that makes efficient use of human effort involves reasoning about which tasks to employ and how much effort to devote to each task. These decisions rely on understanding human performance on individual tasks, and on understanding how the probabilistic and possibly erroneous outputs from each task affect the final solution, either directly or through other tasks that take its output as their input.

The crowd's performance on any given task is often imprecisely known *a priori*, and the space of possible algorithms for solving a problem—involving different combinations of tasks and allocations of effort to tasks—is potentially very large. It is often costly if not infeasible for a designer to empirically compare a large number of algorithms, or to conduct a large number of experiments to learn about the crowd's performance on different tasks. In practice, experiments are often conducted on an ad hoc basis, with designers relying mostly on their intuitions and common practices to determine which algorithms to deploy. Even when deployed algorithms effectively coordinate a crowd to solve a problem, they are not necessarily efficient and may not make the best use of human effort.

To enable designers to discover more efficient algorithms and workflows with less experimentation and manual effort, we develop a general framework for *automated workflow synthesis*. Leveraging the active, indirect elicitation framework of automated environment design, we construct models of human performance on tasks for the purpose of improving a workflow. Over repeated interactions, an automated system selects experiments to refine current models, with the intent of quickly discovering an efficient workflow built on (a subset of) tasks that meets desired objectives and satisfies resource constraints.

To learn quickly, we develop a *value of information* based elicitation strategy that at any time chooses which task to experiment on based on which experiment is expected to provide information that best informs the choice of algorithm for solving a problem. This is done by comparing the expected difference in solution quality between the best algorithm generated using current information and algorithms optimized based on refined information that may be learned from experimentation. In order to reason about the effect of human task performance on the overall performance of an algorithm, we develop a simulation-based approach that uses available models to estimate the cost and solution quality associated with an algorithm. This allows us to compare workflows without having to deploy them, and is used for synthesizing the best workflow given currently available knowledge and for deciding which experiments to conduct.

We illustrate the effectiveness of our approach in a case study on *human sorting tasks*, in which human judgment is used to determine the ordering among objects being sorted. We focus on a class of quicksort algorithms in which pivot selection and pairwise comparison tasks are performed by the crowd, and consider the problem of determining how many workers to devote to each task at each level of recursion. Experimental results show that knowledge of crowd performance on tasks allows us to better optimize for algorithms that are tailored to the crowd and the designer's objective. Results also show that our elicitation strategy reveals better algorithms more quickly than selecting experiments to uniformly reduce uncertainty across models.

Section 8.1 reviews related work. Section 8.2 provides a model of the automated workflow synthesis problem. Section 8.3 introduces a general approach for automated workflow synthesis based on the active, indirect elicitation framework of automated environment design. We introduce a simulation-based approach for evaluating algorithms and present an elicitation strategy that refines current knowledge by selecting experiments to maximize the expected value of information. Section 8.4 describes the human sorting task. We introduce models for predicting human performance on pivot selection and pairwise comparison tasks, and provide a local search procedure for synthesizing sorting workflows. Section 8.5 presents experimental results. Section 8.6 discusses a number of possible extensions and directions for future work.

8.1 Related Work

A number of studies in human computation have developed optimization procedures and control strategies for enabling more efficient computation with humans and machines. For example, Shahaf and Horvitz [84] studied generalized task markets with human and machine problem solvers, and introduced formulations for optimally assigning and sequencing tasks to humans and machines to maximize the utility derived from the final solution. While we also optimize workflows by reasoning about effective combinations of tasks, we consider simultaneously the problem of learning about human performance on tasks. In addition, by utilizing simulations and local search algorithms, we are able to handle optimization problems over complex workflows in which the quantitative relationship between the crowd's performance on tasks and the quality of the final solution is difficult to capture analytically.

Drawing on techniques from decision-theoretic planning, Dai et al. [16, 17] introduced a framework for optimizing workflows by controlling at run-time the request for additional work (e.g., for the purpose of redundancy) based on costs and the inferred work quality. Recent work by Lin et al. [57] showed that a similar approach can be used to dynamically switch between workflows, which can sometimes lead to improvements over using a single workflow. In these works, the structure of the workflow or the set of workflows considered is predetermined and the goal is to efficiently control the computation given fixed designs. In contrast, our framework for automated workflow synthesis aims to tackle the complementary problem of discovering efficient designs in the first place by optimizing over the space of *possible* workflows, which determines the overall structure of the optimized algorithm and the allocation of effort within.

A number of studies have focused on enabling efficient human computation in the context of human-powered database systems that recruit a crowd to perform operations such as filters, sorts, and joins. For example, Marcus et al. [63, 62] introduced a declarative workflow engine called Qurk, and proposed optimizations for sorts and joins such as batching tasks, using numerical ratings, and pre-filtering tables before joins. Venetis et al. [94] studied human computation algorithms for retrieving the maximum item from a set, and proposed a framework for selecting algorithm parameters to optimize the tradeoff over quality, monetary cost, and execution time. By exploring the space of possible algorithms and providing performance models and optimization procedures, findings from these studies can be utilized within an automated workflow synthesis framework to help identify efficient crowd-tailored algorithms for these and related problems.

In machine computation, *program synthesis* considers the use of appropriate design tactics to systematically derive a program based on a problem specification. In the context of sorting, Darlington [18] and Smith [86] demonstrated how to derive a number of sorting algorithms using logical transformations and reductions. Closer to our work, Li et al. [55] demonstrated how to synthesize sorting algorithms that are optimized for particular computer architectures. As learning about crowd abilities incurs a cost, our work on synthesizing sorting algorithms for the crowd must tackle the added challenge of learning quickly, to synthesize efficient algorithms after few experiments.

From the machine learning perspective, our value of information based elicitation strategy can be viewed as taking an *active learning* approach to acquiring information. In the context of human sorting tasks, Pfeiffer et al. [72] introduced an algorithm that adaptively selects which pairwise comparison questions to ask a crowd in order to quickly derive an accurate aggregate ranking using the crowd's noisy answers. While in both this work and our work on synthesizing sorting algorithms the goal is to learn quickly and make efficient use of human effort, we consider through automated workflow synthesis different ways through which humans can contribute to solving a problem. In doing so, we seek to better understand how to structure efficient crowd problem solving by synthesizing algorithms involving heterogeneous tasks.

In artificial intelligence, the study of *metareasoning* [36, 80] focuses on enabling agents with bounded time and computational resources to make intelligent decisions about what to reason about, how long to deliberate for, and when to take action. Since deliberation can lead to better decisions but incurs a cost, it is often necessary to evaluate the benefit and cost of gathering information through additional computation [37, 9, 79]. Due to the cost of human effort, our automated system for workflow synthesis faces a similar problem in that it must decide on which experiments to conduct, how much resources to devote to experimentation, and when to stop experimenting. In using value of information computations to inform elicitation decisions, we adopt a decision-theoretic framework for active, indirect elicitation that draws on principles introduced by Horvitz [35, 36] for decision-theoretic metareasoning.

8.2 Automated Workflow Synthesis

We consider a situation in which an automated system seeks to identify an efficient human computation algorithm or workflow for solving a problem. Given a (potentially large) space of human computation algorithms $\mathcal{A} = \{A_1, \ldots, A_n\}$, we let S_i denote the set of base-level human tasks in A_i that can be assigned directly to individual workers in a crowd,[1] such that $S = S_1 \cup \ldots \cup S_n$ represents the entire set of human tasks under consideration. Each task $s \in S$ is associated with a *task function* f_s, which defines for each task s an output distribution on the space of possible answers, some of which may be incorrect. This captures the distribution over answers that

[1] For example, in the context of Amazon Mechanical Turk, these base-level tasks are the human intelligence tasks (HITs) assigned to workers.

individuals in the crowd may provide when assigned a task. For an algorithm A_i, we let F_i represent an *algorithm function* that maps problem instances into a distribution over solutions. The solution distribution based on F_i is itself constructed from the output distributions of tasks $s \in S_i$, which are based on f_s.

A task encompasses all the details of how the work is requested, which includes for example the user interface and instructions. Two tasks that request the same work may thus produce different distributions over answers. Furthermore, algorithms that share some or all of the same tasks may differ in the type of inputs that are passed to the tasks, and in when and how often each task is called. Algorithms that contain similar or even the same tasks may thus induce different distributions over solutions. Deciding which algorithm to use depends not only on the crowd's performance on tasks, but also on details of the algorithm that govern how outputs combine and propagate to form a final solution.

Given a distribution over problem instances and a measure of the solution quality, the system seeks to identify an algorithm $A^* \in \mathcal{A}$ that achieves a high solution quality on average while satisfying cost constraints.[2] We assume that each instance of a call to a task incurs a known cost, which may be monetary or be based on a measure of the time or effort required to complete the task. In contrast, we assume that the system does not know how well the crowd can perform each task *a priori* (that is, f_s is imprecisely known), and thus cannot perfectly predict the expected quality of solutions obtained through different algorithms.

[2] Our framework is agnostic to details of the objective. We can also consider optimizing for cost subject to constraints on quality or more complex utility-based objectives that define explicit tradeoffs between solution quality and cost.

In order to learn about the crowd's performance on tasks, the system can experiment with different tasks and observe the crowd's outputs. At any time, the system can select from a set of possible experiments $E = \{e_1, \ldots, e_m\}$, each of which corresponds to a particular task-input pair. Since the crowd's answers are probabilistic, the same experiment may result in different observations. For simplicity, we assume that all possible experiments are feasible, such that if an experiment is conducted the corresponding task will be completed by the crowd. A general goal is to quickly discover, after few experiments, an efficient algorithm that obtains high quality solutions and satisfies cost constraints. Since conducting experiments takes time and is also costly, this allows us to deploy better algorithms sooner, and also keeps the cost of experimentation low.[3]

8.3 An Active, Indirect Elicitation Approach

We introduce a general approach for automated workflow synthesis that leverages the active, indirect elicitation framework of automated environment design. For each task $s \in S$, we construct a *task performance model* $\hat{f}_s$ to predict the output from the actual task function f_s. Using observed outputs from experiments, an inference procedure updates $\hat{f}_s$ after each experiment to refine the system's knowledge of the crowd's performance on tasks. This allows the system to better predict the performance of different algorithms under consideration, based on which to optimize the choice of algorithm. In order to select experiments that lead the system to quickly

[3] The elicitation strategy we develop later in this chapter is able to consider explicit tradeoffs between the cost and value derived from experimentation. For simplicity, we do not model the cost of experimentation and focus instead on discovering efficient algorithms quickly.

discover efficient workflows, we introduce a simulation-based approach that allows us to compare different algorithms based on models, and an elicitation strategy that uses simulations to evaluate the value that can be derived from different experiments.

8.3.1 Simulating Human Computation Algorithms

At any point in the active, indirect elicitation process, we assume that the system can use the current task performance model $\hat{f}_s$ for task s to estimate a distribution over outputs for any input to f_s. Under this assumption, the system can simulate an algorithm A_i on a machine by sampling from the output distribution provided by $\hat{f}_s$, $s \in S_i$ whenever the algorithm makes a call to task s. For any algorithm applied to a problem instance, this allows the system to estimate a distribution over possible solutions. Simulations can thus be used to estimate the solution quality for any algorithm based on our current knowledge of the crowd's performance on tasks the algorithm calls upon. Furthermore, since the number of times each task is called in a run of an algorithm may in general depend on the crowd's performance on tasks, simulations also allow us to estimate, to the best of our current knowledge, the cost associated with running an algorithm.

In addition to evaluating algorithms based on current knowledge, we can also use simulations to estimate the solution quality of an algorithm under different hypotheses about f_s. This is helpful when deciding among a set of experiments to conduct.

Having the ability to simulate algorithms tackles two major obstacles for automated workflow synthesis. First, it allows us to compare algorithms without having to necessarily deploy them, which is useful when we are trying to determine which experiment to conduct next. Second, it allows us to reason about complex algorithms that may be difficult to analyze analytically, which makes this approach applicable to a broad range of settings.

8.3.2 Elicitation Strategy

An automated workflow synthesis problem may consider a large space of possible algorithms that draw on diverse tasks. Models for each task may be complex and difficult to learn accurately with few examples. Given this, we would like to be able to determine which experiment to conduct at any time, for which the knowledge acquired may significantly affect our choice of algorithm. Since our goal is ultimately to discover efficient workflows and not to learn about the crowd's performance on tasks, it is not necessary to learn about the task whose model has the most variance, or on which the fewest experiments have been conducted thus far. For example, if we have reason to believe that a task is unlikely to help an algorithm achieve high quality solutions anyway, it is unlikely that learning about this task will provide useful information for improving our choice of algorithm.

Following this intuition, we consider an elicitation strategy that selects experiments based on which task-input pair is most likely to reveal information that improves the choice of the optimal algorithm. Let $A^*_{\hat{f}}$ denote the optimal choice of algorithm to deploy based on current task performance models, such that $A^*_{\hat{f}}$ achieves the highest average solution quality across all algorithms that satisfy cost constraints when simulated using $\hat{f}_s$ for task s on problem instances drawn from a known distribution. For each experiment $e \in E$ that involves the task s_e, let $O_e = \{o^1_e, \ldots, o^k_e\}$

denote the set of potential outcomes from experiment e based on $\hat{f}_{s_e}$. Depending on the realized outcome of an experiment, we may be in one of k possible worlds, corresponding to the state of task performance models after the inference procedure performs an update based on the result of the experiment. We let $\hat{f}^{o_e^i}$ denote the updated task performance models under the assumption that we conduct experiment e and observe outcome o_e^i, and let $A^*_{\hat{f}^{o_e^i}}$ denote the optimal choice of algorithm with respect to $\hat{f}^{o_e^i}$.

Since we can potentially deploy different algorithms based on the outcome of an experiment, the difference in solution quality between $A^*_{\hat{f}}$ and each of the algorithms $A^*_{\hat{f}^{o_e^i}}$, evaluated with respect to our knowledge after observing o_e^i, captures the expected value to be gained if we were to update our choice of algorithm to deploy after conducting a single experiment e. By comparing experiments in this way, we can find the experiment that (myopically) maximizes the expected value of information by solving the following optimization problem:

$$max_{e \in E} \sum_{o_e^i \in O_e} \Pr(o_e^i | \hat{f}_{s_e}) [v(A^*_{\hat{f}^{o_e^i}} | \hat{f}^{o_e^i}) - v(A^*_{\hat{f}} | \hat{f}^{o_e^i})] \tag{8.1}$$

$\Pr(o_e^i|\hat{f})$ is an estimate of the likelihood of observing outcome o_e^i when conducting experiment e based on the task performance model $\hat{f}$, and $v(A|\hat{f})$ is a measure of the expected quality of solutions provided by algorithm A based on task performance model $\hat{f}$.

An elicitation strategy based on this objective focuses experimentation on where there is the most value to be derived from learning. Since an individual experiment only obtains a single output from the crowd, it may not contain enough information to change the decision about the best algorithm. The myopic value of information may be zero for all experiments, but the choice of experiment still matters because conducting an experiment can enable subsequent experiments to become (myopically) valuable. For this reason, it can be important to conduct experiments in batch, where at any given time the elicitation strategy selects a set of experiments to conduct whose potential outcomes best inform the choice of algorithm. Outside of any computational concerns, the elicitation strategy remains essentially the same, but with each experiment representing a set of experiments.

As a technicality, in the context of constrained optimization, an algorithm optimized based on current information may be infeasible in light of information derived from observing the outcome of an experiment. In particular, the term $v(A^*_{\hat{f}}|\hat{f}^{o^i_e})$ may not be well defined. For example, an experiment may reveal that an algorithm that repeatedly calls the same task until multiple solutions agree incurs higher costs than expected if observed outputs are more varied than expected. In these situations, such an algorithm may, by nature of being infeasible, achieve a higher solution quality than an algorithm optimized based on newly derived information. To avoid uninformative comparisons to an infeasible algorithm when making value of information computations, we can apply a "primal heuristic" that transforms an infeasible algorithm into a similar, feasible algorithm. We can then perform any comparisons using the transformed algorithm instead, with the view that the difference in performance between an algorithm optimized based on new information and this transformed algorithm captures the value of information that can be derived from experimentation. Later in the chapter, we construct a primal heuristic for use in the sorting setting we consider.

8.4 Human Sorting Tasks

Having presented a general approach for automated workflow synthesis, we consider as a case study the problem of finding efficient human computation algorithms for *human sorting tasks*. In a human sorting task, human perception and judgment are used to determine the ordering among objects. Examples of human sorting tasks include sorting images by their visual appeal, sorting traffic photos by the severity of traffic conditions presented, sorting edited versions of a paragraph by how well written they are, and sorting web pages by their relevance to a query. Human sorting tasks may vary in their level of objectiveness, but share the common feature that machines often cannot accurately determine the desired ordering among objects.

There are many possible ways to sort, and designing computer algorithms for sorting is of course a well-studied problem. While it is sometimes straightforward to adapt a sorting algorithm for a human sorting task, the effectiveness of the resulting human computation algorithm will depend on how well the crowd can perform the human tasks that the algorithm calls upon. Since people can make mistakes even for objective tasks, solutions may not be perfectly sorted, and redundancy may be needed to achieve good solutions. The algorithm design space thus includes not only different types of sorting algorithms, but also different allocations of effort to tasks within algorithms. Given a constraint on the total cost of effort that can be incurred, the goal is to synthesize a human computation algorithm that maximizes the expected solution quality for an objective of interest.

We focus on the problem of automatically synthesizing a workflow from a class of human computation algorithms based on quicksort, that leverages the crowd to

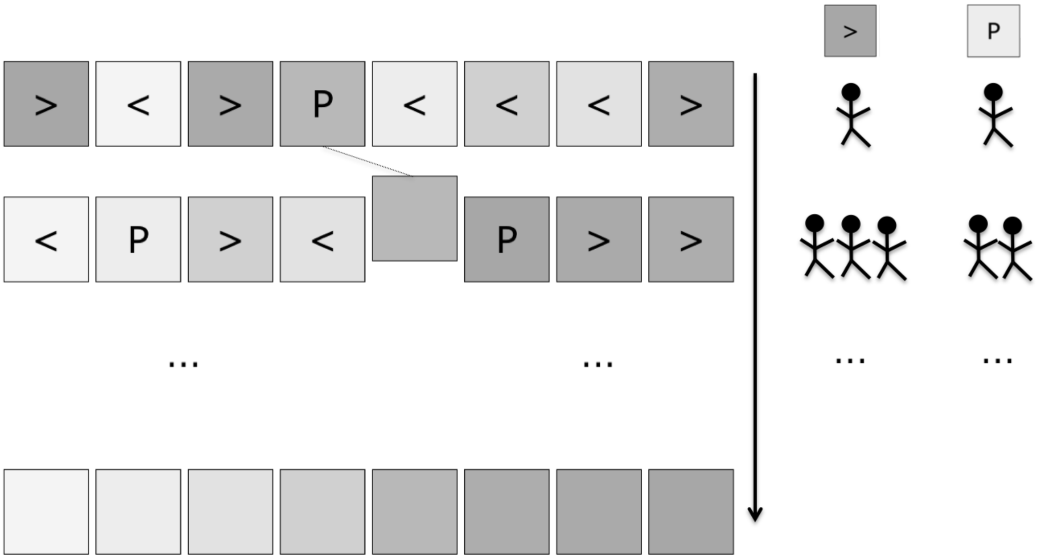


Figure 8.1: The progression of sorting with human quicksort applied to ordering grayscale tiles from light to dark. The algorithm determines the amount of human effort to allocate to each pairwise comparison and pivot selection task at different levels of recursion.

perform pairwise comparison and pivot selection operations. Quicksort is a divide-and-conquer sorting algorithm that sorts a list of elements by first identifying groups of elements that are less than or greater than a *pivot* element, and then recursively applying quicksort on each group. The choice of the pivot affects the algorithm's running time, and for example can be chosen based on the median of three elements selected randomly from the list.

In adapting quicksort for human sorting tasks, we consider how much redundancy to require for each pairwise comparison and pivot selection task that is assigned to the crowd at different points in the computation (see Figure 8.1). These decisions affect the quality of the solution, as well as the number of operations and thus cost required to compute a solution. For example, allocating more effort to pairwise comparisons early in the computation helps to place elements in roughly the correct order, whereas allocating more effort later in the computation increases the likelihood that adjacent elements are in the correct order.

Specifically, we consider optimizing over two sets of parameters r_d and k_d, that determine the number of people to recruit for identifying the median of three randomly chosen elements as the pivot (r_d), and the number of people to recruit for comparing a pair of objects (k_d), at the d-th level of recursion. In cases where $r_d = 0$, a random element is chosen as the pivot. For any task, the algorithm takes the majority answer from people recruited to perform the task as output, breaking ties randomly as needed. Algorithm 8.1 presents the pseudocode for the class of human quicksort algorithms as a function of r_d and k_d, in which MedianOfThree() and PairwiseCompare() represent the pivot selection and pairwise comparison tasks respectively.

The performance of an algorithm in this class depends on how well the crowd can identify the median and perform pairwise comparisons, and on the implications of the crowd's performance on the quality of the solution and the cost incurred. We assume that each call to a pairwise comparison or pivot selection task incurs known costs c_c and c_p respectively, which are additive and independent of the input to a task. To evaluate solution quality, we consider *inversions* as a measure of sortedness. Given a list $\{l_1, \ldots, l_n\}$ that should be sorted in ascending order, the number of inversions is the number of pairwise elements that are out of order, which occurs whenever $l_j < l_i$ for $j > i$. The goal is to find parameter values r_d, k_d such that Human Quicksort based on these values produces solutions with few inversions on average, while staying within a cost budget C.

Algorithm 8.1 Human Quicksort

```
Require: $\{r_d\}$, $\{k_d\}$
procedure HumanQuicksort($\{l_1, \ldots, l_n\}$, $t$)
    if $n = 1$ then
        return $\{l_1\}$
    else if $n = 2$ then
        if PairwiseCompare($l_1, l_2, k_t$) then
            return $\{l_1, l_2\}$
        else
            return $\{l_2, l_1\}$
    else
        $L = \{\}, R = \{\}$
        $p \leftarrow$ MedianOfThree($l, r_t$)
        for $i = 1 \rightarrow n$ do
            if PairwiseCompare($l_i, p, k_t$) then
                Add $l_i$ to $L$
            else
                Add $l_i$ to $R$
        return HumanQuicksort($L, t + 1$) $\cdot \{p\} \cdot$ HumanQuicksort($R, t + 1$)
```

8.4.1 Task Performance Models

In order to discover efficient algorithms quickly, we apply our framework for automated workflow synthesis to sorting by first constructing models for the pairwise comparison and pivot selection tasks. Given two distinct objects a and b, a pairwise comparison task outputs the correct answer with some probability p, and the incorrect answer with probability $1 - p$. The probability of error may depend on aspects of the task such as its user interface and instructions, on how "close" a and b are, etc. Given three elements a, b, and c, the median-of-three pivot selection task outputs the median element with probability q, the smallest element with probability p, and the largest element with probability $1 - p - q$. Similarly, the probability of error may depend on aspects of the task, the relative closeness of a, b, and c, and so on.

While we do not have access to the actual task functions and thus do not know these probabilities *a priori*, we can construct a probabilistic task performance model as follows. Since it is infeasible to learn probabilities for every combination of input values separately, we consider grouping sets of input values into *clusters*, and learning a model for each task-cluster pair. In the simplest instantiation, there may only be a single cluster per task, and the model may only attempt to learn an input-independent probability distribution over outputs. We can consider arbitrarily more complex models by considering finer-grained clusters.

For each model, we use Beta and Dirichlet distributions to represent our knowledge and uncertainty over the actual output distributions for pairwise comparison and pivot selection tasks, respectively. Distributions can capture any prior knowledge we may have about human performance on each task and be updated based on observations from experiments.[4] With Beta and Dirichlet distributions, we can incorporate observations from experiments by simply updating the corresponding model's parameters based on a worker's output. For example, for pairwise comparisons, a

[4] For simplicity, we treat each model as independent, and only perform updates on a model whose cluster matches the inputs to the task in an experiment.

Beta distribution's α and β parameters can capture the number of correct and incorrect answers respectively, and be updated by incrementing α by 1 if the answer to an experiment is correct or incrementing β by 1 otherwise. For pivot selections, a Dirichlet distribution with three parameters maintains counts over the frequency of the correct output and the two possible incorrect outputs (for each cluster), and can be similarly updated.

8.4.2 Simulating Algorithms

Each task performance model maintains a distribution over the actual output distribution for the task. As we conduct more experiments, a model becomes more certain about the crowd's performance on the task and thus allows us to more accurately predict the performance of algorithms. To measure the performance of an algorithm using current models, we can sample using the current task performance models probability distributions over the possible outputs to each task. Each sample represents a "guess" of the probability distribution over outputs based on the actual task function. For each sample, we can simulate the algorithm using the sample as the task function, and obtain a distribution over possible solutions. By aggregating results across samples, we can obtain a "best guess" over the distribution of possible solutions based on current knowledge of crowd performance on tasks as captured by our models.

In order to compute the value of information that can be derived from selecting an experiment, our elicitation strategy needs to be able to simulate algorithms with respect to hypothetically refined models that incorporate updates based on outcomes that may be observed from an experiment. To do so, we can sample output distributions using hypothetically refined task performance models whose Beta and Dirichlet parameters have been updated to take into account possible observed outcomes, but otherwise simulate an algorithm as we would when using current models.

8.4.3 Optimizing Quicksort Algorithms

Our elicitation strategy evaluates the expected value of information that can be derived from conducting an experiment by computing the difference in expected solution quality between (a) the optimal algorithm with respect to current models and (b) the optimal algorithms with respect to information derived from the experiment. For the class of quicksort algorithms we consider, the number of possible algorithms is exponential in the assignment of effort to tasks at different levels of recursion. Computing the optimal algorithm exactly is thus likely to be intractable. To avoid potential computational difficulties, we take a heuristic approach and focus on finding and comparing algorithms that are approximately optimal with respect to task performance models. We do this by adapting for our setting the local search procedures introduced by Venetis et al. [94] for optimizing human computation algorithms for finding the maximum element in a set.

To perform our search, we assume that there is a fixed, finite set of possible values to assign to parameters r_d and k_d. Given task performance models, we first compute the optimal *constant sequence* algorithm, which selects fixed values for r^* and d^* such that $r_d = r^*$ and $k_d = k^*$ for all recursion levels d. Since the space of such algorithms is small, we can obtain the algorithm that maximizes solution quality while satisfying the cost constraint by simply simulating and evaluating every possible constant sequence algorithm within the class.

The optimal constant sequence algorithm serves as a starting point for our search. Better algorithms may exist that allocate effort non-uniformly at different levels of recursion. Fixing the number of people to assign to pivot selection tasks (r_d), we consider a hill-climbing procedure that iteratively varies the number of people to assign to pairwise comparison tasks (k_d). For every pair of parameter values k_i and k_j for which $k_i > 1$, we consider the effect of decrementing k_i and incrementing k_j up to the point that the resulting algorithm *just* satisfies cost constraints. If any such swaps improve the solution quality, we apply the best such swap, and repeat the process to incrementally improve the choice of algorithm until no such improvements exist.[5]

8.4.4 Applying the Elicitation Strategy

While this local search procedure may not necessarily find the optimal algorithm with respect to a set of task performance models, we can nevertheless use the algorithms it produces to evaluate the value of information that can be gained from an experiment. Since individual experiments may not contain enough information to affect the choice of algorithm, we only consider batch experiments which obtain multiple observations at once. Assuming that the set of experiments to consider is

[5] We can construct a generalized local search procedure that considers all possible constant values $r_d = r$. We can also allow for varying values of r_d by fixing the values for r_d and k_d one level of recursion at a time. To do this, given fixed values $k_1, \ldots, k_{i-1}$ and $r_1, \ldots, r_{i-1}$, we identify the values k_i and r_i based on the solution of the generalized local search procedure applied to searching over values of r_d and k_d that have yet to be fixed. See Venetis et al. [94].

not too large, we can compute the expected value of information for each experiment and select the experiment with the largest expected value.

As discussed at the end of Section 8.3.2, one problem we may encounter in making value of information computations is that a human quicksort algorithm optimized based on current information may be infeasible in light of information derived from observing the outcome of an experiment. In particular, the term $v(A^*_{\hat{f}}|\hat{f}^{o^i_e})$ may not be well defined. When this occurs, an algorithm optimized based on current information may appear to be better (by nature of being infeasible) than an algorithm optimized based on an experiment's outcome. To avoid uninformative comparisons to an infeasible algorithm and to evaluate the value of an experiment even in such situations, we apply a "primal heuristic" that makes an infeasible human quicksort algorithm feasible by reducing the number of people it assigns to some of the tasks. We do this by iteratively decrementing some pairwise comparison parameter k_d until the algorithm becomes feasible. At each step, we select a parameter to decrement that leads to the largest (myopic) decrease in cost incurred per unit decrease in solution quality. This procedure seeks to identify a version of the original algorithm that has similar performance but does not violate cost constraints when evaluated based on hypothetically refined performance models. In this way, the difference in solution quality between the optimal algorithm given refined information and the transformed algorithm still represents the value gained when reoptimizing the choice of algorithm based on new information derived from an experiment.

8.5 Experiments

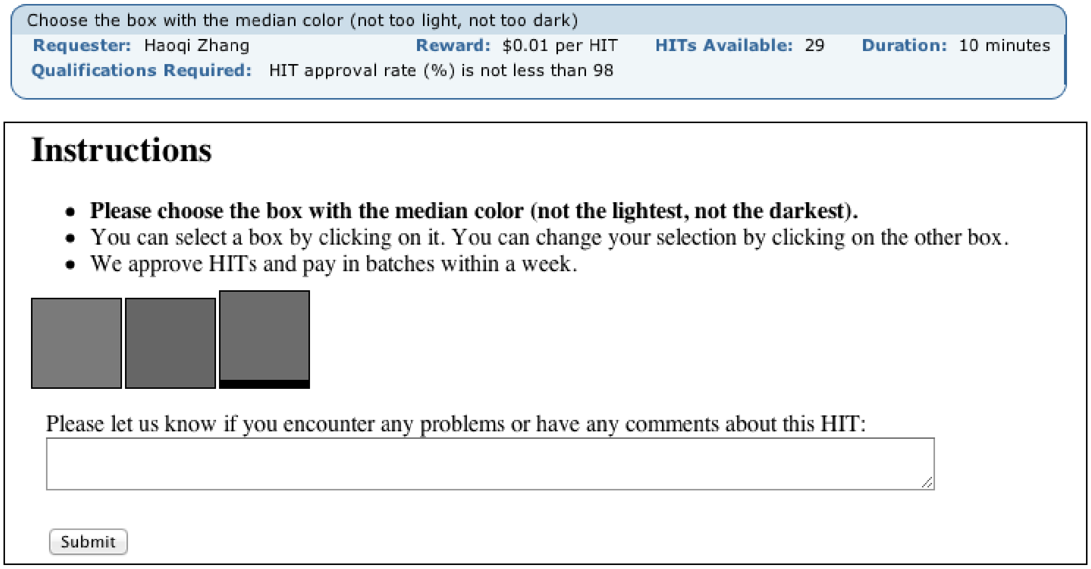
To test the effectiveness of our approach on human sorting tasks, we consider experiments for learning task performance models and optimizing human quicksort algorithms. We focus on two main questions: (a) does learning task performance models help to discover more efficient algorithms, and (b) does our value of information based elicitation strategy lead to more efficient algorithms more quickly than a simple elicitation strategy?

8.5.1 Setup

We consider a human sorting task in which the goal is to sort a list of grayscale tiles from light to dark. We chose this domain because comparisons are objective, tasks are easy to describe, and tasks may vary in difficulty (e.g., depending on how close tiles are in their grayscale value). This makes it easier for us to evaluate answers, increases the likelihood that workers understand the goal of the task, and allows for interesting models that depend on characteristics of particular task instances.

To understand human performance on this task, we recruited workers from Amazon Mechanical Turk (Turkers) to complete pairwise comparison and median-of-three pivot selection tasks. For pairwise comparison tasks, we posted 100 HITs and requested 10 assignments for each HIT. We sampled pairs of grayscale values for tiles at random, restricting the difference in value to between 1 and 10.[6] For pivot selection tasks, we also posted 100 HITs each with 10 assignments. We sampled three

[6] We used a scale with 128 values, such that black is 0 and white is 127. We chose this scale over a 256 valued scale so that minimal differences in darkness are barely distinguishable.



Choose the box with the median color (not too light, not too dark)

Requester: Haoqi Zhang **Reward:** $0.01 per HIT **HITs Available:** 29 **Duration:** 10 minutes

Qualifications Required: HIT approval rate (%) is not less than 98

Instructions

- **Please choose the box with the median color (not the lightest, not the darkest).**
- You can select a box by clicking on it. You can change your selection by clicking on the other box.
- We approve HITs and pay in batches within a week.

Please let us know if you encounter any problems or have any comments about this HIT:

Figure 8.2: An example HIT of the pivot selection task.

grayscale values for tiles at random, restricting the difference in value between the median element and the other 2 elements to between 1 and 10. Workers were required to have a 98% approval rating, and were paid $0.01 per HIT. Figure 8.2 shows an example HIT of the pivot selection task.

To simplify our evaluation, we use the Turkers' responses to construct *ground truth models* of task functions that provide distributions over answers to tasks based on the empirically observed answers from the crowd. When evaluating the active, indirect elicitation approach, instead of actually posting jobs on Mechanical Turk for experiments an elicitation strategy chooses, we instead sample from the ground truth distribution to simulate the answers the crowd would provide in an experiment. Assuming that models are accurate, results of the simulation experiments would still be indicative of the crowd's actual performance, but with the evidence obtained a priori to allow for a simpler evaluation.

For both the ground truth models and the task performance models, we cluster inputs to tasks based on the closeness of the objects being compared, according to our hypothesis that tiles are more difficult to compare when their grayscale values are closer. For pairwise comparison tasks, we consider clusters that correspond to different distances in grayscale value between pairs of objects. For pivot selection tasks, we consider clusters based on the *minimum* distance between the grayscale value of the median element and any non-median element. For both tasks, we consider five clusters each, for distances of 1, 2, 3, 4, and 5+. The models for the pivot selection task maintain counts or probabilities for three possible outcomes: (1) the median is selected, (2) the element closer to the median is selected, and (3) the element farther from the median is selected.[7] We hypothesize that if some of the elements being compared are very close together, people are more likely to make mistakes in favor of the element closer to the median than the element farther from it.

In the active, indirect elicitation process, we maintain a model for each task-cluster pair, which also forms the set of experiments that we can conduct at any given time.[8] We batch experiments to sets of five observations each, such that any update to a model is based on five outcomes drawn from the ground truth distribution for the task-cluster pair. To evaluate the value of information based elicitation strategy, we compare it to a *uniform* strategy that chooses the next experiment based on whichever model has been experimented on the fewest times thus far. We hypothesize that

[7] Whenever two non-median elements are equidistant to the median, a model for the pivot selection task chooses between them with equal probability whenever the median is not chosen.

[8] Pairwise comparison models are initialized with $\alpha = 4$ and $\beta = 1$. Pivot selection models are initialized with $\alpha_1 = 6$, $\alpha_2 = 1$, and $\alpha_3 = 1$.

	Pairwise		Pivot		
Difference	Pr(correct)	Pr(incorrect)	Pr(median)	Pr(closer)	Pr(farther)
1	0.74	0.26	0.585	0.27	0.145
2	0.87	0.13	0.69	0.16	0.15
3	0.94	0.06	0.74	0.13	0.13
4	0.98	0.02	0.82	0.10	0.08
≥5	0.997	0.003	0.85	0.08	0.07

Table 8.1: Ground truth models based on Turkers' performance on pairwise comparison and pivot selection tasks as a function of the (minimum) difference in grayscale value between tiles.

(regardless of elicitation strategy) learning will lead to better algorithms, but that the value of information elicitation strategy will lead to better algorithms after fewer experiments.

When synthesizing human quicksort algorithms, we consider optimizing with respect to random permutations of a list with 20 tiles holding grayscale values 1 through 20, with costs $c_c = c_p = 1$ and budget $C = 250$.[9] We consider $k_d \in \{1, 3, 5, 7\}$ and $r_d = r \in \{0, 1, 3\}$ as the possible values to assign to parameters k_d and r_d, where $d \in \{1, 2, 3, 4, 5, 6+\}$.

8.5.2 Results

From the Mechanical Turk experiment, we found that people indeed make more mistakes in pairwise comparison tasks when tiles are closer in grayscale value. We observe from Table 8.1 that when tiles only differ in value by 1, the crowd makes twice as many mistakes (26% error rate) as when tiles differ in value by 2 (13%), and

[9] Note that since our models only consider the difference in grayscale value between tiles, the exact grayscale values we assign to tiles are inconsequential for the purposes of our experiments.

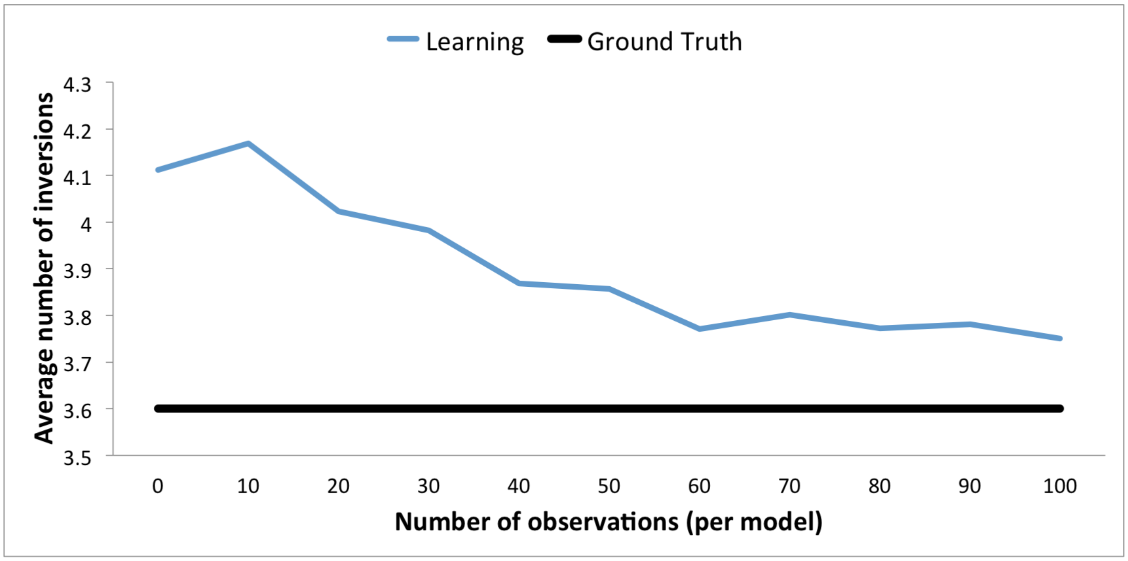


Figure 8.3: The performance (with respect to ground truth models) of algorithms optimized through the process of learning. Values represent averages over 50 trials.

about four times as many mistakes as when tiles differ in value by 3 (6%). The crowd makes very few mistakes for any larger differences in value, which suggests that after a certain point the tiles are noticeably different. For pivot selection tasks, we also found that people make more mistakes when one or more of the non-median elements is close to the median. We observe that when a non-median element is very close to the median (i.e., differ in grayscale value by 1), people are much more likely to make mistakes in favor of selecting that element than the farther non-median element.

Based on workers' answers, we constructed ground truth models using the empirically observed probabilities for each task-cluster pair (Table 8.1). Figure 8.3 shows that the average performance of the algorithm optimized using current task performance models (evaluated with respect to the ground truth models) improves over time as we conduct more experiments in simulation and observe more samples drawn from

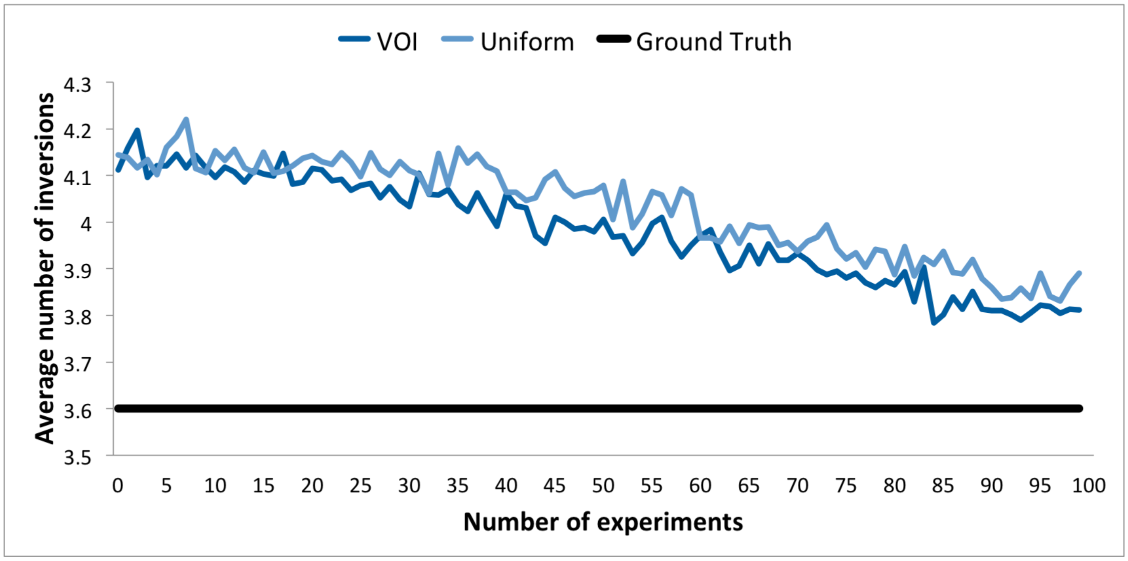


Figure 8.4: Comparison of the performance (with respect to ground truth models) of algorithms optimized based on current models through the process of learning for the value of information (VOI) and uniform elicitation strategies. Values represent averages over 50 trials.

the ground truth distribution.[10] This demonstrates that knowledge acquired from experiments reduces noise and uncertainty in task performance models and informs better decisions when synthesizing workflows based on learned models.

Figure 8.4 compares the average performance of algorithms optimized using current models over the course of learning based on the value of information and uniform elicitation strategies. We observe that for both strategies, solution quality generally improves as more information is collected from experiments. Comparing the two strategies, we observe that at any given point in time, algorithms optimized based on information obtained using the value of information elicitation strategy tend to

[10] As with making value of information computations, we may encounter scenarios in which an optimized algorithm using current task performance models does not satisfy cost constraints with respect to the ground truth distribution. In these cases we apply the primal heuristic earlier discussed and evaluate the performance of the feasible, transformed algorithm instead.

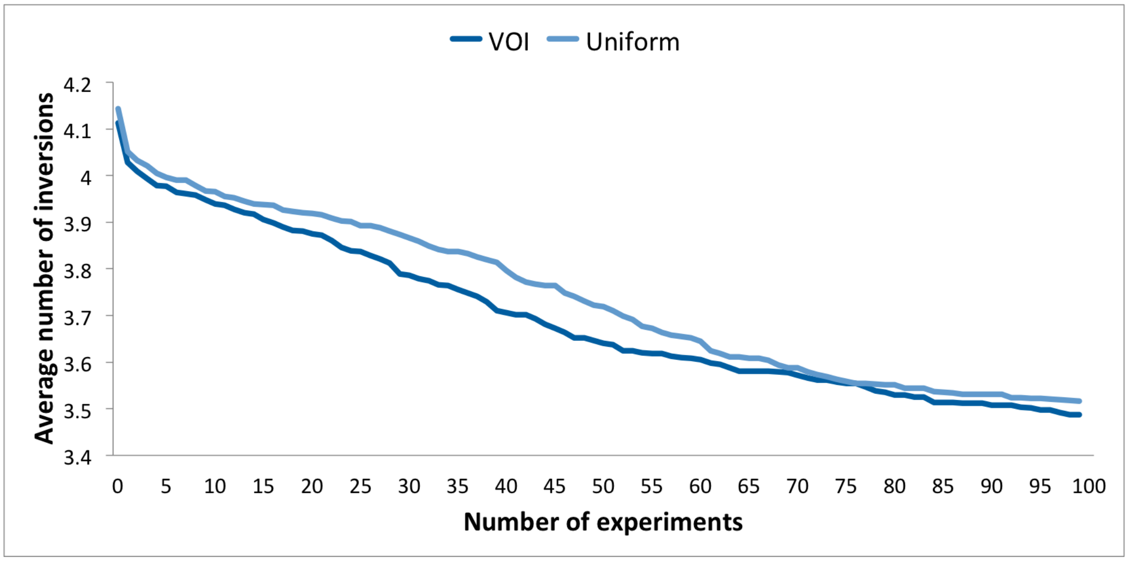


Figure 8.5: Comparison of the performance (with respect to ground truth models) of the best algorithms discovered thus far through the process of learning for the value of information (VOI) and uniform elicitation strategies. Values represent averages over 50 trials.

outperform algorithms optimized based on information obtained using the uniform elicitation strategy (90% of the time). That is, given the same amount of experimentation, the value of information elicitation strategy allows the system to synthesize better algorithms on average than the system can synthesize based on information derived from following the uniform strategy. Viewed differently, for any desired solution quality, the value of information elicitation strategy allows the system to optimize for algorithms achieving that solution quality after fewer experiments.

Since learned task performance models are inherently probabilistic and noisy, there is no guarantee that a piece of evidence obtained from experimentation will necessarily lead to an optimized algorithm with strictly better performance. An algorithm optimized based on current models thus serves as a best guess of what may be a good

	$k_1, k_2, k_3, k_4, k_5, k_{6+}$	r	Inversions	Cost
Best discovered through learning	$3, 3, 5, 3, 7, 5$	1	3.45	249
Synthesized based on ground truth	$3, 3, 3, 5, 7, 5$	3	3.60	248

Table 8.2: Configuration and performance (with respect to ground truth models) of the best human quicksort algorithm discovered through our learning experiments and the human quicksort algorithm synthesized with respect to the ground truth models. We consider random permutations of a list with 20 elements holding values 1 through 20, with costs $c_c = c_p = 1$ and budget $C = 250$.

algorithm. From the designer's perspective, algorithms synthesized at any point in time can be viewed as candidates for A/B testing against the best algorithm discovered thus far that is (presumably) currently deployed. Taking this view, we also compared the value of information elicitation strategy against the uniform elicitation strategy based on the solution quality of the best algorithm discovered thus far. Figure 8.5 shows that on average, the value of information elicitation strategy discovers efficient algorithms more quickly, and at any point in time, has already discovered a more efficient algorithm than has been discovered by the uniform strategy.

Table 8.2 shows the configuration and performance (with respect to ground truth models) of the best human quicksort algorithm discovered through our learning experiments and the human quicksort algorithm synthesized with respect to the ground truth models. We see that both algorithms apply more effort at deeper levels of recursion than at shallow levels (1 and 2). In quicksort, at deeper levels of recursion, any two tiles being compared are more likely to be closer in grayscale value. Since workers are more likely to make mistakes when tiles are close in grayscale value, the additional effort being applied at deeper levels of recursion reduces the likelihood of such errors and thus effectively reduces the number of inversions.

We also observe from Table 8.2 that it is possible for an algorithm synthesized based on learned models to outperform an algorithm synthesized based on ground truth models when evaluated with respect to the ground truth models. This counter-intuitive situation may occur because the local search procedure we use to synthesize algorithms may constrain the search space differently depending on the models considered. In particular, since the local search procedure fixes the number of repetitions (r) used for pivot selection tasks based on the best constant sequence algorithm, depending on the models considered, some values for r are not explored. While the search space with respect to the ground truth models can only consider human quicksort algorithms for which $r = 3$, the search space with respect to learned models may consider different values of r and thus include better algorithms.

In comparing the best algorithm discovered through our learning experiments with the algorithm synthesized based on ground truth models, we observe that with $r = 1$, the best algorithm discovered allocates more repetitions to k_3 and fewer repetitions to k_4, which is infeasible for $r = 3$. This helps to reduce the number of inversions because the nominal number of calls to pairwise comparison tasks (not counting how many repetitions are requested for each task) for which the tiles' grayscale values are 1 apart is highest at level 3. This is due to the recursive structure of quicksort. Since there are roughly twice as many lists at level 4 than at level 3, there are roughly twice as many pivots selected at level 4. Since tiles being compared against the pivot are those that have yet to be selected as a pivot, the number of pairwise comparison tasks decreases rapidly as we move to deeper levels of recursion. While the fraction of pairwise comparison tasks for which grayscale values are 1 apart is higher at level

4, at level 3 there is significantly more pairwise comparison tasks. The effort shifted from repetitions assigned to pivot selection tasks and to k_4 is thus put to good use through k_3 to reduce the potential for error in more pairwise comparisons tasks for which the likelihood of error is highest.

8.6 Discussion

We applied the active, indirect elicitation framework of automated environment design to the problem of automated workflow synthesis and demonstrated how learning about human performance on tasks and synthesizing workflows based on learned models can enable more efficient human computation. To discover more efficient algorithms more quickly, we introduced an elicitation strategy that reasons about the value of information that can be derived from conducting different experiments and focuses the learning on where this value is greatest. Results from experiments on human sorting tasks showed that the elicitation strategy is effective for quickly discovering efficient algorithms that are tailored to the crowd's performance on tasks.

Our framework and methods are quite general, and can be extended in a number of ways. In the context of sorting, we can for example consider a larger set of possible tasks beyond pairwise comparison and pivot selection, and include in the design space other classes of sorting algorithms beyond quicksort. As some tasks may be used in multiple algorithms, any knowledge of the crowd's performance on such a task will inform the design of all algorithms that use it. In addition to deciding how to allocate effort within each class of algorithms, we can also consider optimizing over *hybrid sort algorithms*. For example, we can consider using one algorithm to first order items roughly and another algorithm to then refine the sort, which may be more efficient in some settings [62].

We saw from observing Turkers' performance when comparing grayscale tiles that task performance can depend on not only the task, but on specifics of the problem instance. In general, for accurately predicting the crowd's performance in order to effectively synthesize algorithms, models of task performance may need to be quite rich. This suggests that a model may require significant effort and domain knowledge to construct and a significant amount of data from experiments to learn. Given that the same tasks may be used in not only different algorithms for solving a particular problem, but also in different contexts for solving completely different problems, we would like to be able to reuse designer and crowd effort by building extendable libraries of task performance models that can be reused in other automated workflow synthesis problems. Such libraries would allow for a "warm start," where one can begin reasoning about algorithms using already learned information about some tasks and focus learning efforts on other tasks. Learned models can similarly be incorporated into libraries for future reuse.

While we have focused primarily on the learning problem, considering a more complex design space brings into focus computational challenges in optimizing and synthesizing workflows. Since synthesizing an algorithm may involve choosing tasks and allocating effort to tasks, both of which are combinatorial in general, the problem can be arbitrarily hard computationally. Having tractable procedures that can effectively search over the design space and discover efficient algorithms quickly is crucial, both for the purpose of quickly deploying designs based on learned information and for making decisions based on elicitation strategies that synthesize workflows under different knowledge conditions as subroutines in value of information calculations. As in active, indirect elicitation approaches more generally, synthesis procedures can use current models of participant behavior to help constrain the search for a well-tailored design.

Human computation algorithms may include tasks for machine computation. The performance and efficiency of such tasks can be similarly measured, modeled, and reasoned about when synthesizing workflows. Our framework extends straightforwardly to include machine tasks, and allows for learning and optimizing over human-machine algorithm design spaces. The decision-making over whether to use human or machine computation components may consider particular tradeoffs in efficiency, cost, and performance [84].

As mentioned in our discussion of related work, we make a conceptual distinction between automated workflow synthesis and decision-theoretic control [16, 17]. Workflow synthesis is about *algorithm design*, and focuses on reasoning about the structure of an algorithm before it is deployed. Decision theoretic control is about *execution control*, and focuses on reasoning about the state and progression of problem solving in the midst of solving a problem. As design and control both influence eventual performance and complement one another, future work should explore considering both aspects in unison, which may lead to discovering new techniques and approaches.

Chapter 9

Conclusion

The Internet today is a center for social and economic activity. Through crowdsourcing, social media, and electronic commerce, social and economic systems on the Internet attract large numbers of individuals to take action and join in collaborations. From a system designer's perspective, a key challenge is understanding how to promote particular desired participant behaviors and outcomes. I call this problem *computational environment design*.

The designer's role is to construct the *decision environment* in which participants take actions. This can include interfaces, workflows, feedback to users, incentives, constraints on actions, rules and policies, and so forth. Participants have their own preferences and capabilities, that together with the decision environment influence their behavior. As the designer can only affect participants' actions and outcomes indirectly through the decision environment, solving computational environment design problems may rely on understanding participants and tailoring designs to the participants.

In this dissertation, I propose an approach for solving computational environment design problems by reasoning and learning about characteristics of participants and how these characteristics interact with the decision environment to influence behavior. By reasoning, I mean thinking about participants and a design problem using available knowledge. By learning, I mean the acquisition of new knowledge about participants that informs design decisions.

I focus on two notable abilities afforded by the Internet that speak directly to the computational environment design problem. The first is the ability to recruit a crowd. Taking advantage of this ability, *crowdsourcing* and *human computation* systems are attracting crowds to solve large-scale problems. From a computational environment design perspective, this presents an exciting opportunity for designers to recruit large numbers of interested participants for the *explicit purpose* of performing useful actions that help to achieve desired outcomes.

A practical challenge that arises when attracting a large crowd to perform an arbitrary task is that individuals may only be briefly involved, and any given individual may provide noisy inputs. Leveraging a crowd to complete a complex task may thus require coordinating small, noisy contributions from large numbers of participants, or identifying and attracting individuals who are most willing and able to contribute.

In the first part of the dissertation, I show how reasoning about crowd abilities and limitations can lead to designs that enable a crowd to effectively contribute to solving complex tasks. In seeking to leverage the *distributed intelligence* of the crowd, I make advances in three core directions. The first direction is the coordination among problem solvers. I demonstrate how existing design patterns can be effectively combined to construct human computation algorithms and new design patterns that enable the crowd to solve complex problems (Chapters 2). I also introduce a *crowdware* design, that enables the crowd to tackle complex tasks involving global constraints which cannot be easily decomposed and solved using human computation algorithms (Chapter 3).

The second direction is harnessing the *general intelligence* of the crowd. I explore methods and designs that engage the crowd to guide the control flow of an algorithm and generate plans that define the problem-solving process (Chapter 4). In studying effective means for passing solution context in the 8-puzzle and a system called CrowdPlan for generating simple plans to high level search queries, we are beginning to explore principles for *crowdsourcing general computation* that can enable general problem solving via human computation systems.

The third direction is the recruitment of expertise. I study *task routing* as an approach for problem solving in which individuals both contribute to a solution and route to others for further contributions (Chapter 5). Focusing on prediction tasks, I introduce incentive mechanisms that promote participants to honestly report private information and route tasks to people who they believe can best contribute.

In the process of arriving at effective designs, I find that designs that are effective for small groups of people are not necessarily effective for the crowd. Such designs often needed to be rethought and adapted to explicitly take into account crowd abilities and limitations. For example, in studying Mobi, we observed how automatically generated *todo items* are crucial for helping the crowd keep track of violated constraints in the process of generating an itinerary. Given crowd workers who are only briefly involved and not available over time to keep track of solution context, the todo items take on the role of a "dedicated volunteer," who is there always to provide feedback and make suggestions about how to move toward a solution.

As another example, when constructing *routing scoring rules* for task routing for prediction tasks, we observed that it is possible to construct incentives such that if everyone knows about the network structure and everyone else's expertise, then in equilibrium everyone would route along the optimal path. But in social networks on the Internet, individuals may only know (the expertise of) people within their local neighborhood, which may only include their friends and possibly friends of friends. Implementing the would-be optimal incentive scheme would require people to perform complicated inference and may not work as desired. As a solution, we introduced a class of *local routing rules*, that are designed to explicitly enable equilibrium behavior for which the inference required of participants is local and thus tractable.

The second ability afforded by the Internet with implications for computational environment design is the ability for designers to engage in a data-driven, iterative design process. The Internet provides a wide range of tools for iterative design, that include web analytics software for tracking user behavior; style sheets, frameworks, and content management systems for redesigning easily; and tools for A/B testing for comparing designs based on desired objectives. From a computational environment design perspective, these tools provide a valuable resource for designers of Internet systems to easily experiment with alternative designs, collect rich behavioral data from large numbers of users, and iterate quickly to improve designs over time.

But despite having these tools, the process of designing for effective behavior on the Internet remains largely manual, tedious, and ad hoc. Experiments are often conducted on alternative designs that consist of small modifications aimed at making incremental improvements against set objectives. Without particular regard to understanding participants and their behavior, this can lead to a design process that hill-climbs toward a solution at a local rather than global maximum. Designers may miss out on better designs, and ultimately fail to promote desired behaviors and outcomes.

In the second part of the dissertation, I introduce principles and methods that enable an automated system to systematically explore a design space to elicit desired behavior by reasoning and learning about participants. In *automated environment design*, a system takes a model of the interaction among decision environment, participants, and behaviors and seeks to quickly identify an effective intervention from a space of possible interventions. We introduce an *active, indirect elicitation* framework that drives an objective-based, iterative design process (Chapter 6). The framework makes use of an *inference procedure* and an *elicitation strategy.* The inference procedure uses observations of participant actions to learn about participants and refine existing models of behavior. The elicitation strategy complements the inference procedure by designing experiments to refine existing knowledge.

We find through applications to crowdsourcing that an automated system using observations of participant actions to refine a model of behavior can discover effective designs tailored to the participants that achieve significantly better outcomes than designs available prior to learning. In *automated task design*, we learned models of the quality of worker output to an image labeling task as a function of task design parameters and used learned models to construct optimized designs that are demonstrated to be more effective at the same rate of pay (Chapter 7). In *automated workflow synthesis*, we learned models of the crowd's performance on pairwise comparison and pivot selection tasks and used learned models to better allocate effort within a human quicksort algorithm to achieve high quality solutions (Chapter 8).

In general, a design space may be very large, and exploring it blindly may not lead to effective designs. By learning about participants from observing their behavior in response to different designs, we can effectively narrow the space of possible designs we need to consider. This is because our knowledge of participants gives us a better sense of which designs may be effective or ineffective. In the extreme case where we have a perfect, known model of how participants make decisions with respect to different designs, identifying the best design becomes an optimization problem with known parameters. When studying the problem of *policy teaching*, we take advantage of this insight and develop a centroid-based elicitation strategy that is guaranteed to elicit the desired behavior after few interactions (Chapter 6). The elicitation strategy does this by basing incentives on hypotheses that, if correct, will elicit the desired behavior, and if incorrect will lead to an observation that significantly narrows the space of agent rewards that are consistent with observed behavior.

In addition to exploring a design space in a principled manner based on models of participant behavior, to be practically useful, automated environment design systems need to be able to discover effective designs quickly. In the context of computational environment design, the goal is not to learn about participants for learning's sake but rather to elicit desired behaviors and outcomes quickly. Focusing on this, in studying automated workflow synthesis, we introduced a *value of information* based elicitation strategy that selects experiments by estimating the expected value that can be derived from potential improvements to the current choice of algorithm as the result of new information (Chapter 8). By incorporating the objective of the designer directly into the elicitation strategy, a value of information approach focuses the learning effort on exploring parts of the design space where learning is most likely to matter.

In both manual and automated approaches to solving computational environment design problems, reasoning and learning about participants allows us to discover effective solutions that are tailored to the participants. In the rest of the chapter, we briefly review the main contributions and results, and present directions for future work.

9.1 Brief Review

The first part of the dissertation focused on human computation and crowdsourcing and introduced design patterns and methods for recruiting and coordinating a crowd to tackle complex tasks.

Chapter 2 studied the design of human computation algorithms that enable the crowd to contribute effectively to complex tasks. Through the problem of crowdsourcing audio transcription, I discovered an *iterative dual pathway structure* that effectively combines the output-agreement design pattern with the iterative design pattern to encourage contributors to provide accurate improvements. This design pattern eliminates the need for explicit quality control via voting or grading and focuses the crowd's effort on improving solutions instead. I then considered the problem of crowdsourcing nutrition analysis from food photographs. I introduced a system called PlateMate, whose workflow consists of multiple, heterogeneous tasks that request from the crowd diverse contributions such as tagging food items, describing ingredients, and measuring portions. PlateMate is built on the management framework inspired by the structure of human organizations, which provides effective support for managing complex workflows involving heterogeneous tasks.

Chapter 3 presented a crowdware design that enables a crowd to tackle complex tasks with global constraints through a shared, collaborative workspace. Focusing on crowd itinerary planning as a case study, I introduced a system called Mobi. Mobi presents a single interface through which individuals in the crowd can see the current solution and all ideas generated thus far and contribute freely. To guide the crowd towards useful contributions, Mobi displays automatically generated todo items that alert crowd workers of unresolved constraints. The design takes advantage of the crowd's ability to process context and contribute where they are best able to. It also addresses the crowd's limited attention span by bringing to their attention via todo items where contributions are most needed. Experiments and user studies showed that the design is effective in helping workers to resolve global constraints and that the crowd-generated itineraries satisfied users' stated mission requirements.

In crowdware and Mobi, the crowd is allowed to shape the problem solving process directly. That is, the process of computation is no longer fixed by an algorithm ahead of time and is instead defined by the crowd in the process of problem solving. Expanding on this view, Chapter 4 explored opportunities for involving the crowd in control and synthesis. I presented an experiment on the 8-puzzle that demonstrated how passing a small amount of context can enable more effective problem solving in an iterative task. I also introduced a system called CrowdPlan, that leveraged a crowd to generate simple plans that help users to approach and accomplish high-level tasks.

Solving a complex problem requires not only effective coordination but recruiting individuals who are willing and able to effectively contribute. Chapter 5 proposed methods for task routing on a social network that harness people's ability to both contribute to a solution and route tasks based on their knowledge of others' expertise. Focusing on prediction tasks, I introduced routing scoring rules that properly incentivize participants to honestly update probability assessments and route tasks to people who they believe can best contribute. Taking into account that individuals may only know about people within a local neighborhood, I identified a family of local routing rules which isolate simple routing decisions in equilibria while still promoting effective information aggregation.

Understanding participants and their behavior is crucial for designing any social or economic Internet system that aims to elicit desired behaviors and outcomes. To enable designers to discover more effective designs more quickly and with less manual effort, the second part of this dissertation focused on constructing automated procedures that discover effective designs by reasoning and learning about participants.

Chapter 6 introduced a general approach for automated environment design. I provided a model of the automated environment design problem and presented an active, indirect elicitation framework that drives an objective-based, iterative design process. As an illustrative example, I introduced the problem of policy teaching, in which the goal is to discover rewards that induce an agent to follow a desired policy. I showed that, even with a large number of possible designs and little prior information about the agent's reward function, an algorithm based on the active, indirect elicitation framework is guaranteed to discover an effective reward intervention after a small number of interactions.

Chapter 7 presented an approach for automating the design of human computation tasks. Using image labeling as an example, I learned models of the crowd's performance by observing the crowd's outputs under different task designs and used learned models to optimize designs based on desired objectives. Experimental results showed that simple models can accurately predict work quality and that optimized designs outperformed baseline designs at the same rate of pay.

While Chapter 7 focused on the design of human computation tasks with identical, parallel subtasks, Chapter 8 considered the more general challenge of automating the synthesis of workflows that involve heterogeneous tasks. By adapting the active, indirect elicitation framework of automated environment design, I introduced a general framework for automated workflow synthesis. I presented an elicitation strategy that decides which task to experiment on at any given point by estimating the expected value that can be derived from new information. Learned models are used to synthesize and tune algorithms to optimize desired objectives subject to resource constraints. In experiments on human sorting tasks, I showed that this elicitation strategy is effective in helping to discover better algorithms with less experimentation.

9.2 Research Directions

9.2.1 Crowdsourcing and Human Computation

In crowdsourcing complex tasks, there are opportunities to develop other crowdware systems along with theoretical models, in order to fundamentally understand the spectrum between crowdware and workflow paradigms. An interesting challenge is scale. As the number of ideas and the size of the solution grows, it becomes difficult if not impossible for any given individual to keep track of the entire solution context and reason about all aspects of the problem. For problems that are difficult to decompose, managing problem-solving context becomes difficult and crucial for effective problem solving. Problems embodying this challenge include enabling a crowd to write a novel or a large piece of software, and involving hundreds or thousands of individuals in planning real world events and executing their plans.

One idea for overcoming the challenge of scale is to present solution context at different levels of detail and abstraction. We can create *task platforms* that generalize both crowdware and workflow paradigms. By presenting context at the right level of detail, individuals can be prompted to make effective local contributions while being aware of the effect of their actions on the global solution. For example, in writing a story, someone working on the plot may need to be aware of the impact of his contributions on character development, but can otherwise contribute freely. In cases where relevant views of the solution may not already exist, such views may need to be explicitly constructed by the crowd to facilitate effective problem solving. For example, a crowd writing a story may need to produce plot summaries and character profiles to help people working on different aspects of the problem be aware of relevant changes that require attention and thus be able to contribute more effectively.

From the workflow perspective, task platforms are algorithms that determine and display at any given time a set of available tasks, and for any selection construct an interface that provides the necessary context and functionality for that task. From the crowdware perspective, task platforms consist of multiple workspaces that cover different but interdependent aspects of the problem. There are opportunities to explore both perspectives, and to develop frameworks, methods, and applications that leverage this concept.

Moving from the task level to the organizational level, we can envision a future in which the *distributed intelligence* of humans and machines across networks are brought together to tackle complex problems. In the context of task routing, there are opportunities to develop general principles and methods that effectively and efficiently harness the diverse expertise of participants in a system. As online labor markets and online platforms for collaborative problem solving [2, 84] develop, it will become increasingly important to make efficient use of people's expertise. This includes recognizing people's changing levels of attention, motivation, and availability, and the corresponding need for balancing the load across participants. For example, a task should not always be routed to the individual with the most expertise, simply because that individual may already be engaged in another task. As effective problem solving may rely on the *joint* characteristics of participants involved, I am interested in exploring settings in which *ad hoc teams* [89] of human and machine problem solvers connected through networks form spontaneously to tackle problems as they arise, and expect that reasoning and learning about the *collective intelligence* [101] of such teams may affect solutions and outcomes.

As we continue to explore crowd problem solving in the context of complex and creative tasks, we will inevitably encounter or create scenarios in which the crowd has an intrinsic interest in the solution. In other words, the outcome of the crowd's collaboration may matter to the crowd, and this creates new opportunities and challenges. One possible issue that can arise is that while some individuals may only be briefly involved, other individuals may return to a task over time and claim particular tasks as their responsibility or make demands about some aspect of the solution. For example, an individual contributing to writing a story may attempt to steer the plot toward a certain direction, and individuals planning a large-scale event may not agree on the best course of action.

One approach for resolving differences in opinion and making key decisions is to consider differentiating members within a crowd, such that some contributors may be given special powers and privileges based on their experience or expertise, and can serve as moderators or decision makers should conflicts arise. This is common on the Web, and is used in social computing systems like Wikipedia and in forums to resolve disputes and maintain the quality of content. In the case of a crowd, a hierarchy among contributors may emerge either organically or based on rules and policies set by the designer. For example, within a task platform, it is possible that some tasks are at a higher level than other tasks (e.g., decision about a key aspect of a plot), with these tasks only accessible to those who have already contributed significantly to other tasks within the platform. Understanding how to design such rules and policies in order to create and maintain cultural norms through which the best contributors can emerge organically is an interesting area for future work.

An alternative approach is to design affordances that promote effective crowd decision making, but otherwise leave the decision to the crowd. For example, to settle differences, members in the crowd may vote on the best path forward, with the system automatically enforcing and imposing that choice unless the results from a subsequent vote suggests a different path forward. In the context of planning a real-world event, this may mean voting on a course of action and sticking to it unless the crowd collectively prefers something else. The crowd can also decide to split up into smaller crowds, each pursuing their own direction forward. From the computational environment design perspective, understanding how to design effective mechanisms for joint decision making within crowds that promote effective outcomes is an interesting area for future work.

9.2.2 Automated Environment Design

For automated environment design, a key next step is applying the active, indirect elicitation framework to a wide range of real world scenarios in which automation may help to discover more effective designs more quickly and with less manual effort. In the near term, there are opportunities to automate the design of websites and web pages to promote desired usage patterns, for example to increase the levels of contribution, comprehension, and awareness. In the longer term, there may be opportunities to apply automated design techniques in the physical world, where advances in ubiquitous sensing and the increasing digitization of real world spaces have the potential to enable interactions in which spaces can automatically configure themselves to promote desired behaviors and outcomes.

An ongoing challenge for automated environment design will be the availability of models that capture how characteristics of participants interact with the decision environment to influence participant behavior. But as improved models and computational tools for understanding participants from data become available, automated environment design tools and methods will naturally play an increasingly important role in how we approach the design of social and economic systems.

Of course, human ingenuity will also continue to play an important role in designing social and economic systems for many years to come. An interesting direction is to explore opportunities for tight-knit collaboration between humans and machines in the process of identifying an effective design for solving a computational environment design problem. As an example, consider the following interaction. An automated system forms hypotheses and suggests experiments on alternative designs on its own. A designer can at any time ask questions about how the process is going, provide feedback by identifying particular neighborhoods to focus the search, and introduce additional features and parameters for the system to incorporate in its automated design process. The system may likewise provide feedback on a designer's hypotheses and make suggestions based on its knowledge. While the example may seem somewhat futuristic, given the extent to which automated tools for simplifying the design of social and economic systems on the Internet are already utilized and continue to be developed, such interactions may not be so far fetched, and point to a future in which effective collaboration among human and machine designers become commonplace.

9.3 One More Thing

One can envision a future in which a crowd is more connected, more intelligent, and generally more capable of handling a situation or task than an individual. One can also envision a future in which automated systems are more adept at understanding us and shaping the environment around us.

But there is one more thing I want to discuss. It is about why social and economic systems on the Internet exist in the first place.

A computational environment design problem is intrinsically a human problem. It's about designers with their own interests and motivations constructing decision environments in which participants with their own interests and motivations take action. The environment exists to advance the interests of both parties. Otherwise, a designer would likely modify the environment he controls or participants would leave and new environments would likely form.

Given this, what is perhaps most important is for designers to adopt a way of thinking in which truly advancing the interests of both the designer and the participants is paramount over any narrower objective that can be formed. Without regard to this way of thinking, designers may be content with constructing environments that lead to desired behaviors in the short term but that are ultimately unsustainable. For this reason, a designer may need to continuously reassess specific objectives and designs to ensure that they indeed advance the interests of both parties. Such awareness will require that we develop the ability to reason and learn about fundamentally what it is that we as designers aim to do, why is it that we do what we do, and whether doing what we do indeed makes things better.

Bibliography

[1] Lada A. Adamic, Rajan M. Lukose, Amit R. Puniyani, and Bernardo A. Huberman. Search in power-law networks. *Physical Review E*, 64:046135, Sep 2001.

[2] Salman Ahmad, Alexis Battle, Zahan Malkani, and Sepander Kamvar. The jabberwocky programming environment for structured social computing. In *Proceedings of the 24th annual ACM symposium on User interface software and technology*, UIST '11, pages 53–64, New York, NY, USA, 2011. ACM.

[3] Bethany Ann Yon, Rachel K. Johnson, Jean Harvey-Berino, and Beth C. Gold. The use of a personal digital assistant for dietary self-monitoring does not improve the validity of self-reports of energy intake. *Journal of the American Dietetic Association*, 106(8):1256–1259, 2006.

[4] Esteban Arcaute, Adam Kirsch, Ravi Kumar, David Liben-Nowell, and Sergei Vassilvitskii. On threshold behavior in query incentive networks. In *Proceedings of the 8th ACM conference on Electronic commerce*, pages 66–74, 2007.

[5] Michael S. Bernstein, Greg Little, Robert C. Miller, Bjoern Hartmann, Mark S. Ackerman, David R. Karger, David Crowell, and Katrina Panovich. Soylent: a word processor with a crowd inside. In *Proceedings of the 23nd annual ACM symposium on User interface software and technology*, UIST '10, pages 313–322. ACM, 2010.

[6] Dimitris Bertsimas and Santosh Vempala. Solving convex programs by random walks. *Journal of the ACM*, 51(4):540–556, 2004.

[7] J. Eric Bickel. Some comparisons among quadratic, spherical, and logarithmic scoring rules. *Decision Analysis*, 4:49–65, June 2007.

[8] Fatima Boujarwah, Gregory Abowd, and Rosa Arriaga. Socially computed scripts to support social problem solving skills. In *Proceedings of the 2012 ACM annual conference on Human Factors in Computing Systems*, CHI '12, pages 1987–1996, 2012.

[9] Jack S. Breese and Eric J. Horvitz. Ideal reformulation of belief networks. In *Proceedings of the Sixth Annual Conference on Uncertainty in Artificial Intelligence*, UAI '90, pages 129–144, New York, NY, USA, 1991. Elsevier Science Inc.

[10] Catherine M. Champagne, George A. Bray, April A. Kurtz, Josefina B.R. Monteiro, Elizabeth Tucker, Julia Volaufova, and James P. Delany. Energy intake and energy expenditure:: A controlled study comparing dietitians and non-dietitians. *Journal of the American Dietetic Association*, 102(10):1428–1432, 2002.

[11] Dana Chandler and John Horton. Labor allocation in paid crowdsourcing: Experimental evidence on positioning, nudges and prices. In *Proceedings of the 3rd Human Computation Workshop*, HCOMP '11, June 2011.

[12] Dana Chandler and Adam Kapelner. Breaking monotony with meaning: Motivation in crowdsourcing markets. Working paper, University of Chicago, 2010.

[13] Li Chen and Pearl Pu. Survey of preference elicitation methods. Technical report, Swiss Federal Institute Of Technology In Lausanne (EPFL), 2004.

[14] Yiling Chen, Daniel M. Reeves, David M. Pennock, Robin D. Hanson, Lance Fortnow, and Rica Gonen. Bluffing and strategic reticence in prediction markets. In *Proceedings of the 3rd international conference on Internet and network economics*, WINE '07, 2007.

[15] S. Cooper, F. Khatib, A. Treuille, J. Barbero, J. Lee, Michael Beenen, A. Leaver-Fay, D. Baker, Z. Popovic, and FoldIt Players. Predicting protein structures with a multiplayer online game. *Nature*, 466:756–760, August 2010.

[16] Peng Dai, Mausam, and Daniel S. Weld. Decision-theoretic control of crowd-sourced workflows. In *Proceedings of the 24th AAAI Conference on Artificial Intelligence*, AAAI '10, pages 1168–1174, 2010.

[17] Peng Dai, Mausam, and Daniel S. Weld. Artificial intelligence for artificial artificial intelligence. In *Proceedings of the 25th AAAI Conference on Artificial Intelligence*, AAAI '11, pages 1153–1159, 2011.

[18] John Darlington. A synthesis of several sorting algorithms. *Acta Informatica*, 11:1–30, 1978. 10.1007/BF00264597.

[19] Devansh Dikshit and Narahari Yadati. Truthful and quality conscious query incentive networks. In *Proceedings of the 5th International Workshop on Internet and Network Economics*, WINE '09, 2009.

[20] Julia M. Dinkins. Beliefs and attitudes of americans towards their diet. US Department of Agriculture Center for Nutrition Policy and Promotion, 2000.

[21] Peter Sheridan Dodds, Roby Muhamad, and Duncan J. Watts. An experimental study of search in global social networks. *Science*, 301(5634):827–829, 2003.

[22] John Douceur and Thomas Moscibroda. Lottery trees: Motivational deployment of networked systems. In *Proceedings of the 2007 conference on Applications, technologies, architectures, and protocols for computer communications*, SIGCOMM '07, 2007.

[23] Steven Dow, Anand Kulkarni, Scott Klemmer, and Björn Hartmann. Shepherding the crowd yields better work. In *Proceedings of the ACM 2012 conference on Computer Supported Cooperative Work*, CSCW '12, pages 1013–1022, New York, NY, USA, 2012. ACM.

[24] Fabio A. Drucker and Lisa K. Fleischer. Simpler sybil-proof mechanisms for multi-level marketing. In *Proceedings of the 13th ACM Conference on Electronic Commerce*, EC '12, pages 441–458, New York, NY, USA, 2012. ACM.

[25] Clarence A. Ellis, Simon J. Gibbs, and Gail Rein. Groupware: some issues and experiences. *Communications of the ACM*, 34(1):38–58, 1991.

[26] Yuval Emek, Ron Karidi, Moshe Tennenholtz, and Aviv Zohar. Mechanisms for multi-level marketing. In *Proceedings of the 12th ACM conference on Electronic Commerce*, EC '11, pages 209–218, New York, NY, USA, 2011. ACM.

[27] Annelies H.C. Goris, Margriet S. Westerterp-Plantenga, and Klaas R. Westerterp. Undereating and underrecording of habitual food intake in obese men: selective underreporting of fat intake. *The American journal of clinical nutrition*, 71(1):130, 2000.

[28] Branko Grunbaum. Partitions of mass-distributions and of convex bodies by hyperplanes. *Pacific Journal of Mathematics*, 10(4):1257–1261, 1960.

[29] Robin Hanson. Logarithmic market scoring rules for modular combinatorial information aggregation. *Journal of Prediction Markets*, 1(1):3–15, February 2007.

[30] Barbara Hayes-Roth and Frederick Hayes-Roth. A cognitive model of planning. *Cognitive Science*, 3:275–310, 1979.

[31] Damon Horowitz and Sepandar D. Kamvar. The anatomy of a large-scale social search engine. In *Proceedings of the 19th international conference on World Wide Web*, WWW '10, pages 431–440, New York, NY, USA, 2010. ACM.

[32] John J. Horton and Lydia B. Chilton. The labor economics of paid crowdsourcing. In *Proceedings of the 11th ACM conference on Electronic commerce*, EC '10, pages 209–218, New York, NY, USA, 2010. ACM.

[33] John J. Horton, David G. Rand, and Richard J. Zeckhauser. The Online Laboratory: Conducting Experiments in a Real Labor Market. *Experimental Economics*, 14(3):399–425, 2011.

[34] Eric J. Horvitz. Problem-solving design: Reasoning about computational value, tradeoffs, and resources. In *Proceedings of the NASA Artificial Intelligence Forum*, pages 26–43, 1987.

[35] Eric J. Horvitz. Reasoning about beliefs and actions under computational resource constraints. In *Proceedings of the Third Workshop on Uncertainty in Artificial Intelligence*, pages 429–444, 1987.

[36] Eric J. Horvitz. *Computation and Action Under Bounded Resources.* Dissertation, Stanford, 1990.

[37] Eric J. Horvitz and Jack S. Breese. Ideal partition of resources for metareasoning. Technical Report KSL-90-26, Stanford University, 1990.

[38] Eric Huang, Haoqi Zhang, David C. Parkes, Krzysztof Gajos, and Yiling Chen. Toward automatic task design: A progress report. In *Proceedings of the 2nd Human Computation Workshop*, HCOMP '10, 2010.

[39] Panagiotis G. Ipeirotis. Demographics of mechanical turk. *CeDER Working Papers*, 2010.

[40] Panagiotis G. Ipeirotis, Foster Provost, and Jing Wang. Quality management on Amazon Mechanical Turk. In *Proceedings of the 2nd Human Computation Workshop*, HCOMP '10, 2010.

[41] Kalervo Järvelin and Jaana Kekäläinen. Cumulated gain-based evaluation of ir techniques. *ACM Trans. Inf. Syst.*, 20:422–446, 2002.

[42] Rosie Jones and Kristina Lisa Klinkner. Beyond the session timeout: Automatic hierarchical segmentation of search topics in query logs. In *Proceedings of the 17th ACM conference on Information and knowledge management*, CIKM '08, 2008.

[43] Ece Kamar, Severin Hacker, and Eric Horvitz. Combining human and machine intelligence in large-scale crowdsourcing. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems - Volume 1*, AAMAS '12, pages 467–474, Richland, SC, 2012. International Foundation for Autonomous Agents and Multiagent Systems.

[44] Ece Kamar, Eric Horvitz, and Chris Meek. Mobile opportunistic commerce: Mechanisms, architecture, and application. In *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems*, AAMAS '08, 2008.

[45] Aniket Kittur. Crowdsourcing, collaboration and creativity. *XRDS*, 17:22–26, December 2010.

[46] Aniket Kittur, Boris Smus, Robert Kraut, and Susheel Khamkar. Crowdforge: Crowdsourcing complex work. In *Proceedings of the 24th annual ACM symposium on User interface software and technology*, UIST '11, 2011.

[47] Jon Kleinberg. Complex networks and decentralized search algorithms. In *Proc. International Congress of Mathematicians*, 2006.

[48] Jon Kleinberg and Prabhakar Raghavan. Query incentive networks. In *Proceedings of the 46th Annual IEEE Symposium on Foundations of Computer Science*, FOCS '05, pages 132–141, Washington, DC, USA, 2005. IEEE Computer Society.

[49] Nicolas Kokkalis, Johannes Huebner, Steven Diamond, Dominic Becker, Michael Chang, Moontae Lee, Florian Schulze, Thomas Koehn, and Scott R. Klemmer. Automatically providing action plans helps people complete tasks. In *Proceedings of the 4th Human Computation Workshop*, HCOMP '12, 2012.

[50] Anand P. Kulkarni, Matthew Can, and Bjoern Hartmann. Turkomatic: automatic recursive task and workflow design for mechanical turk. In *Proceedings of the ACM 2012 conference on Computer Supported Cooperative Work*, CSCW '12, pages 1003–1012, 2012.

[51] Nicolas Lambert, David M. Pennock, and Yoav Shoham. Eliciting properties of probability distributions: the highlights. *SIGecom Exch.*, 7(3):9:1–9:5, November 2008.

[52] Edith Law and Luis von Ahn. *Human Computation.* Morgan & Claypool Publishers, 2011.

[53] Edith Law and Haoqi Zhang. Towards large-scale collaborative planning: Answering high-level search queries using human computation. In *Proceedings of the 25th AAAI Conference on Artificial Intelligence*, AAAI '11, pages 1210–1215, 2011.

[54] Vladimir I. Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. *Soviet Physics Doklady*, 10:707–710, 1966.

[55] Xiaoming Li, Maria J. Garzaran, and David Padua. Optimizing sorting with machine learning algorithms. In *Proceedings of the IEEE International Parallel and Distributed Processing Symposium*, IPDPS '07, march 2007.

[56] Beatrice Liem, Haoqi Zhang, and Yiling Chen. An iterative dual pathway structure for speech-to-text transcription. In *Proceedings of the 3rd Workshop on Human Computation*, HCOMP '11, 2011.

[57] Christopher Lin, Mausam, and Daniel Weld. Dynamically switching between synergistic workflows for crowdsourcing. In *Proceedings of the 26th AAAI Conference on Artificial Intelligence*, AAAI '12, 2012.

[58] Greg Little. *Programming with Human Computation*. Dissertation, MIT, 2011.

[59] Greg Little, Lydia B. Chilton, Max Goldman, and Robert C. Miller. Exploring iterative and parallel human computation processes. In *Proceedings of the ACM SIGKDD Workshop on Human Computation*, HCOMP '10, pages 68–76, New York, NY, USA, 2010. ACM.

[60] Greg Little, Lydia B. Chilton, Max Goldman, and Robert C. Miller. Turkit: human computation algorithms on mechanical turk. In *Proceedings of the 23nd annual ACM symposium on user interface software and technology*, UIST '10, pages 57–66, New York, NY, USA, 2010. ACM.

[61] Edwin A. Locke and Gary P. Latham. Building a practically useful theory of goal setting and task motivation. *American Psychologist*, 57(9):705–717, 2002.

[62] Adam Marcus, Eugene Wu, David Karger, Samuel Madden, and Robert Miller. Human-powered sorts and joins. *Proceedings of the VLDB Endowment*, 5(1):13–24, September 2011.

[63] Adam Marcus, Eugene Wu, David R. Karger, Samuel Madden, and Robert C. Miller. Crowdsourced databases: Query processing with people. In *Proceedings of the 5th Biennial Conference on Innovative Data Systems Research*, CIDR '11, 2011.

[64] Corby K. Martin, Stephen D. Anton, Emily York-Crowe, Leonie K. Heilbronn, Claudia VanSkiver, Leanne M. Redman, Frank L. Greenway, Eric Ravussin, and Donald A. Williamson. Empirical evaluation of the ability to learn a calorie counting system and estimate portion size and food intake. *British Journal of Nutrition*, 98(02):439–444, 2007.

[65] Corby K. Martin, Hongmei Han, Sandra M. Coulon, H. Raymond Allen, Catherine M. Champagne, and Stephen D. Anton. A novel method to remotely measure food intake of free-living individuals in real time: the remote food photography method. *British Journal of Nutrition*, 101(03):446–456, 2009.

[66] Winter Mason and Duncan J. Watts. Financial incentives and the 'Performance of Crowds'. In *Proceedings of the 1st Human Computation Workshop*, HCOMP '09, June 2009.

[67] Dana Nau, Malik Ghallab, and Paolo Traverso. *Automated Planning: Theory & Practice*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2004.

[68] Andrew Y. Ng and Stuart Russell. Algorithms for inverse reinforcement learning. In *Proc. 17th International Conference on Machine Learning*, pages 663–670. Morgan Kaufmann, San Francisco, CA, 2000.

[69] Jon Noronha, Eric Hysen, Haoqi Zhang, and Krzysztof Z. Gajos. Platemate: Crowdsourcing nutrition analysis from food photographs. In *Proceedings of the 24th annual ACM symposium on User interface software and technology*, UIST '11, pages 1–12, 2011.

[70] Aditya Parameswaran, Hyunjung Park, Hector Garcia-Molina, Neoklis Polyzotis, and Jennifer Widom. Deco: Declarative crowdsourcing. In *Proceedings of the 21st ACM International Conference on Information and Knowledge Management*, CIKM '12, 2012.

[71] David C. Parkes. *Iterative Combinatorial Auctions: Achieving Economic and Computational Efficiency*. Dissertation, University of Pennsylvania, 2001.

[72] Thomas Pfeiffer, Xi Alice Gao, Andrew Mao, Yiling Chen, and David G. Rand. Adaptive polling for information aggregation. In *Proceedings of the 26th AAAI Conference on Artificial Intelligence*, AAAI '12, 2012.

[73] Galen Pickard, Wei Pan, Iyad Rahwan, Manuel Cebrian, Riley Crane, Anmol Madan, and Alex Pentland. Time-critical social mobilization. *Science*, 334(6055):509–512, 2011.

[74] Catherine Pikholz, Boyd Swinburn, and Patricia Metcalf. Under-reporting of energy intake in the 1997 national nutrition survey. *The New Zealand Medical Journal*, 117(1202), 2004.

[75] Martin F. Porter. *An algorithm for suffix stripping*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1997.

[76] Martin L. Puterman. *Markov decision processes: Discrete stochastic dynamic programming*. John Wiley & Sons, New York, 1994.

[77] Luis A. Rademacher. Approximating the centroid is hard. In *SCG '07: Proceedings of the twenty-third annual symposium on Computational geometry*, pages 302–305, New York, NY, USA, 2007. ACM.

[78] Jakob Rogstadius, Vassilis Kostakos, Aniket Kittur, Boris Smus, Jim Laredo, and Maja Vukovic. An assessment of intrinsic and extrinsic motivation on task performance in crowdsourcing markets. In *Proceedings of the 5th International AAAI Conference on Weblogs and Social Media*, ICWSM '11, 2011.

[79] Stuart Russell and Eric Wefald. On optimal game-tree search using rational meta-reasoning. In *Proceedings of the 11th international joint conference on Artificial intelligence - Volume 1*, IJCAI'89, pages 334–340, San Francisco, CA, USA, 1989. Morgan Kaufmann Publishers Inc.

[80] Stuart Russell and Eric Wefald. Principles of metareasoning. *Artificial Intelligence*, 49(1-3):361–395, May 1991.

[81] Earl D. Sacerdoti. *A structure for plans and behavior*. Elsevier North-Holland, 1977.

[82] Dale A. Schoeller, Linda G. Bandini, and William H. Dietz. Inaccuracies in self-reported intake identified by comparison with the doubly labelled water method. *Canadian journal of physiology and pharmacology*, 68(7):941, 1990.

[83] Reinhard Selten. Axiomatic characterization of the quadratic scoring rule. *Experimental Economics*, 1(1):43–61, June 1998.

[84] Dafna Shahaf and Eric Horvitz. Generalized task markets for human and machine computation. In *Proceedings of the 24th AAAI Conference on Artificial Intelligence*, AAAI '10, pages 986–993, 2010.

[85] Aaron D. Shaw, John J. Horton, and Daniel L. Chen. Designing incentives for inexpert human raters. In *Proceedings of the ACM 2011 conference on Computer supported cooperative work*, CSCW '11, pages 275–284, New York, NY, USA, 2011. ACM.

[86] Douglas R. Smith. Top-down synthesis of divide-and-conquer algorithms. *Artificial Intelligence*, 27,1:43–96, 1985.

[87] Rion Snow, Brendan O'Connor, Daniel Jurafsky, and Andrew Y. Ng. Cheap and fast - but is it good? evaluating non-expert annotations for natural language tasks. In *Proceedings of the 2008 Conference on Empirical Methods in Natural Language Processing*, EMNLP '08, pages 254–263, October 2008.

[88] Mark Stefik, Danny G. Bobrow, Gregg Foster, Stan Lanning, and Deborah Tatar. WYSIWIS revised: early experiences with multiuser interfaces. *ACM Trans. Inf. Syst.*, 5:147–167, April 1987.

[89] Peter Stone, Gal A. Kaminka, Sarit Kraus, and Jeffrey S. Rosenschein. Ad hoc autonomous agent teams: Collaboration without pre-coordination. In *Proceedings of the 24th AAAI Conference on Artificial Intelligence*, AAAI '10, 2011.

[90] Qi Su, Dmitry Pavlov, Jyh-Herng Chou, and Wendell C. Baker. Internet-scale collection of human-reviewed data. In *Proceedings of the 16th international conference on World Wide Web*, WWW '07, May 2007.

[91] James Surowiecki. *The wisdom of crowds: why the many are smarter than the few and how collective wisdom shapes business, economies, societies, and nations*. Doubleday, 2004.

[92] Jeffrey Travers and Stanley Milgram. An experimental study of the small world problem. *Sociometry*, 32:425–443, 1969.

[93] Robert J. Vanderbei. *Linear programming : foundations and extensions*. Springer, 3rd edition, 2008.

[94] Petros Venetis, Hector Garcia-Molina, Kerui Huang, and Neoklis Polyzotis. Max algorithms in crowdsourcing environments. In *Proceedings of the 21st international conference on World Wide Web*, WWW '12, pages 989–998, New York, NY, USA, 2012. ACM.

[95] Luis von Ahn. *Human Computation*. PhD thesis, Carnegie Mellon University, 2005.

[96] Luis von Ahn and Laura Dabbish. Labeling images with a computer game. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '04, pages 319–326, 2004.

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

[98] Luis von Ahn, Benjamin Maurer, Colin McMillen, David Abraham, and Manuel Blum. reCAPTCHA: Human-based character recognition via web security measures. *Science*, pages 1465–1468, September 2008.

[99] Duncan J. Watts, Peter Sheridan Dodds, and M. E. J. Newman. Identity and search in social networks. *Science*, 296(5571):1302–1305, 2002.

[100] Duncan J. Watts and Steven H. Strogatz. Collective dynamics of "small-world" networks. *Nature*, 393(6684):440–442, June 1998.

[101] Anita Williams Woolley, Christopher F. Chabris, Alex Pentland, Nada Hashmi, and Thomas W. Malone. Evidence for a collective intelligence factor in the performance of human groups. *Science*, 330(6004):686–688, 2010.

[102] Bethany A. Yon, Rachel K. Johnson, Jean Harvey-Berino, Beth C. Gold, and Alan B. Howard. Personal digital assistants are comparable to traditional diaries for dietary self-monitoring during a weight loss program. *Journal of behavioral medicine*, 30(2):165–175, 2007.

[103] Haoqi Zhang, Yiling Chen, and David C. Parkes. A general approach to environment design with one agent. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence*, IJCAI '09, 2009.

[104] Haoqi Zhang, Eric Horvitz, Yiling Chen, and David C. Parkes. Task routing for prediction tasks. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems*, AAMAS '12, pages 889–896, 2012.

[105] Haoqi Zhang, Eric Horvitz, Rob Miller, and David C. Parkes. Crowdsourcing general computation. Technical Report MSR-TR-2011-6, Microsoft Research, 2011.

[106] Haoqi Zhang, Edith Law, Rob Miller, Krzysztof Gajos, David C. Parkes, and Eric Horvitz. Human computation tasks with global constraints. In *Proceedings of the 2012 ACM annual conference on Human Factors in Computing Systems*, CHI '12, pages 217–226, New York, NY, USA, 2012. ACM.

[107] Haoqi Zhang and David C. Parkes. Value-based policy teaching with active indirect elicitation. In *Proceedings of the 23rd National Conference on Artificial Intelligence*, AAAI '08, pages 208–214, 2008.

[108] Haoqi Zhang, David C. Parkes, and Yiling Chen. Policy teaching through reward function learning. In *Proceedings of the 10th ACM Conference on Electronic Commerce*, EC '09, pages 295–304, 2009.