Crowd-Powered Systems

by

Michael Scott Bernstein

S.M., Massachusetts Institute of Technology, 2008
B.S., Stanford University, 2006

Submitted to the Department of
Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2012

Author . . . .
Department of
Electrical Engineering and Computer Science
May 23, 2012

Certified by . . . .
David R. Karger
Professor
Thesis Supervisor

Certified by . . . .
Robert C. Miller
Associate Professor
Thesis Supervisor

Accepted by . . . .
Leslie A. Kolodziejski
Chair, EECS Committee on Graduate Students

Crowd-Powered Systems

by

Michael Scott Bernstein

Submitted to the Department of
Electrical Engineering and Computer Science
on May 23, 2012, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

Abstract

Crowd-powered systems combine computation with human intelligence, drawn from large groups of people connecting and coordinating online. These hybrid systems enable applications and experiences that neither crowds nor computation could support alone.

Unfortunately, crowd work is error-prone and slow, making it difficult to incorporate crowds as first-order building blocks in software systems. I introduce computational techniques that decompose complex tasks into simpler, verifiable steps to improve quality, and optimize work to return results in seconds. These techniques develop crowdsourcing as a platform so that it is reliable and responsive enough to be used in interactive systems.

This thesis develops these ideas through a series of crowd-powered systems. The first, Soylent, is a word processor that uses paid micro-contributions to aid writing tasks such as text shortening and proofreading. Using Soylent is like having access to an entire editorial staff as you write. The second system, Adrenaline, is a camera that uses crowds to help amateur photographers capture the exact right moment for a photo. It finds the best smile and catches subjects in mid-air jumps, all in realtime. Moving beyond generic knowledge and paid crowds, I introduce techniques to motivate a social network that has specific expertise, and techniques to data mine crowd activity traces in support of a large number of uncommon user goals.

These systems point to a future where social and crowd intelligence are central elements of interaction, software, and computation.

Thesis Supervisor: David R. Karger
Title: Professor

Thesis Supervisor: Robert C. Miller
Title: Associate Professor

Acknowledgments

This thesis is dedicated to a crowd of indispensable individuals:

- my wife Adi, who was also thanked in my undergraduate and masters theses, and who deserves the appreciation even more in this one;
- my parents Neila and Andy and my sister Corinne, who know more about computers than they admit;
- my advisors David Karger and Rob Miller, always willing to listen to crazy ideas;
- Terry Winograd, Scott Klemmer, Björn Hartmann, and Jeff Shrager, who got me into this business in the first place;
- my mentors and role models, Mark Ackerman, Desney Tan, and Ed Chi;
- more colleagues and collaborators than I should have probably enlisted as a graduate student — Adam Marcus, Andrés Monroy-Hernández, Bongwon Suh, Drew Harry, Greg Little, Greg Vargas, Eric Horvitz, Hiroshi Ishii, Jaime Teevan, Jilin Chen, Joel Brandt, Katrina Panovich, Kurt Luther, Lichan Hong, Mary Czerwinski, Max Van Kleek, mc schraefel, Paul André, Sanjay Kairam, Susan Dumais, and Xiao Xiao;
- the community and friends in the Haystack and User Interface Design groups at MIT's Computer Science and Artificial Intelligence Laboratory;
- and the participants on Amazon Mechanical Turk and other crowdsourcing platforms, who both make this research possible and keep it interesting.

I was supported by a National Science Foundation Graduate Research Fellowship, a Microsoft Research PhD Fellowship, a Xerox Fellowship, and some generous advisors. These projects were supported in part by NSF grant IIS-1111044. You all make graduate school possible — thanks.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

As far back as Doug Engelbart's NLS [50] and Ivan Sutherland's Sketchpad [188], human-computer interaction has been structured around a tradeoff between user control and system automation. The user has deep context and understanding of the task, but limited memory and speed. The system has limited context and understanding, but powerful algorithmic resources. As a result, where the system cannot succeed reliably, the user steps in to guide the process manually.

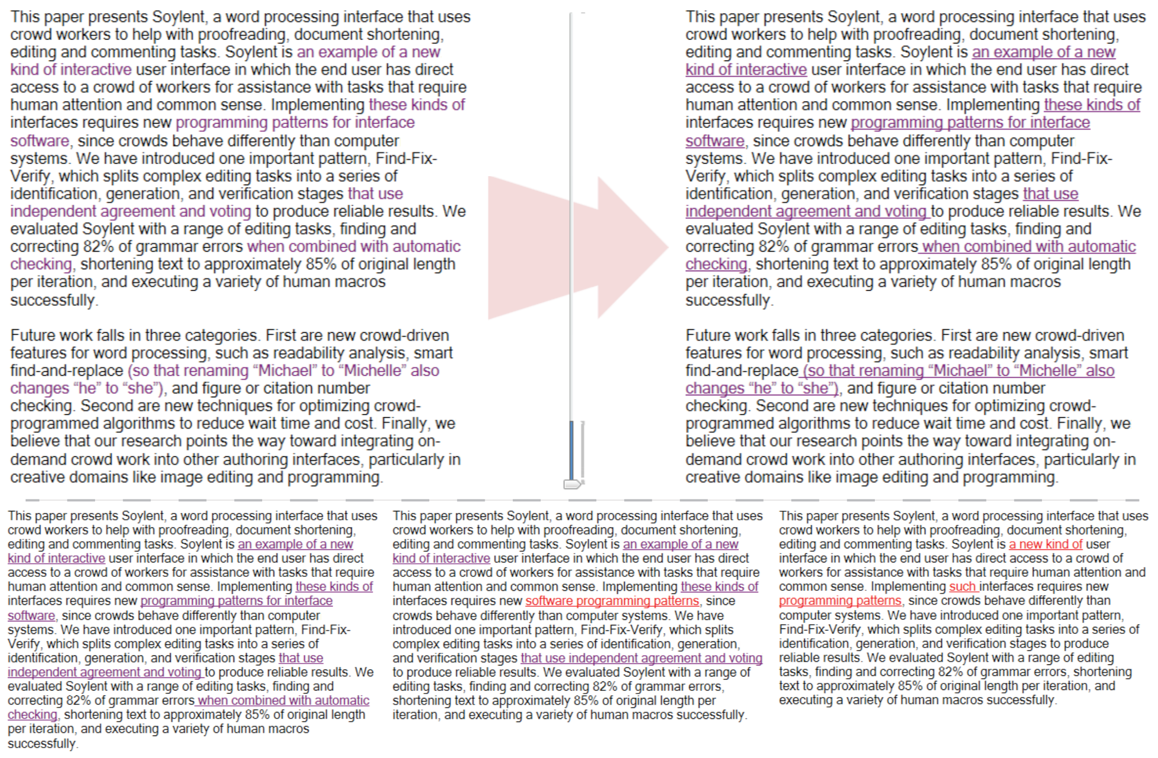
This tradeoff between the system and the user both defines and limits human-computer interaction. If the system cannot reliably support a task or goal, and if the user is unwilling to do it manually, the user may abandon the system. As a result, most interactive systems only attempt tasks they can safely automate. For example, consider the word processor — likely one of the most heavily-used and heavily-designed interactive systems of all time. While it supports secondary tasks like layout and spelling, the word processor has limited support for the primary task it was designed for: writing. Issues such as expressiveness, clarity, and cutting text are beyond the abilities of the system, and thus left entirely to the user.

This thesis proposes that combining computing with the intelligence of *crowds* — large groups of people connecting and coordinating online — allows the creation of hybrid human-computer systems that overcome the limits of the user-system tradeoff. These *crowd-powered systems* do not only rely on the cognitive abilities of the user, but also reach out to the aggregate knowledge, cognition, and perception abilities of many individuals. These crowds may be paid, incentivized through social interaction, or mined from activity traces. This thesis will demonstrate that crowd-powered systems let us re-envision classic interactive computing systems such as the word processor and digital camera. I will also describe how computation can help guide crowds to complete more complex tasks, faster. If successful, this research agenda will impact areas of computing ranging from hybrid crowd-artificial intelligence systems [94, 24, 62] and open-world databases [56, 137, 157] to the broader social science and management disciplines [140].

While large groups are increasingly adept at completing straightforward parallel tasks [187, 67, 95], they can struggle with complex work. Participants vary in quality, well-intentioned contributions often introduce errors, and errors are amplified as they propagate through the crowd. The resulting output is typically poor or incomplete. Without higher-level coordination, many tasks are beyond the ability of crowds today. Moreover, crowds that can generate the necessary information might not even exist yet, or their knowledge might be distributed across the web.

This dissertation develops computational techniques for high-quality, low-latency crowdsourcing. First, the *Find-Fix-Verify* design pattern decomposes complex tasks into simpler, verifiable steps. Second, the *retainer model* of recruitment returns on-demand human results in seconds. Third, we introduce a queueing theory model to optimize the correct crowd size given task requirements. Fourth, we incentivize the collection of specific information needs that generic crowds cannot support. These contributions are critical to the creation of crowd-powered systems: they advance crowdsourcing from a batch platform to one that is interactive and realtime.

The core idea of this thesis is a set of architectural and interaction patterns for integrating crowdsourced human contributions directly into interactive systems. The thesis accomplishes this goal by developing a series of interactive prototypes, each of which advance crowdsourcing further as a platform to support high-quality, fast, or personally-targeted interactivity.


This paper presents Soylent, a word processing interface that uses crowd workers to help with proofreading, document shortening, editing and commenting tasks. Soylent is an example of a new kind of interactive user interface in which the end user has direct access to a crowd of workers for assistance with tasks that require human attention and common sense. Implementing these kinds of interfaces requires new programming patterns for interface software, since crowds behave differently than computer systems. We have introduced one important pattern, Find-Fix-Verify, which splits complex editing tasks into a series of identification, generation, and verification stages that use independent agreement and voting to produce reliable results. We evaluated Soylent with a range of editing tasks, finding and correcting 82% of grammar errors when combined with automatic checking, shortening text to approximately 85% of original length per iteration, and executing a variety of human macros successfully.

Future work falls in three categories. First are new crowd-driven features for word processing, such as readability analysis, smart find-and-replace (so that renaming "Michael" to "Michelle" also changes "he" to "she"), and figure or citation number checking. Second are new techniques for optimizing crowd-programmed algorithms to reduce wait time and cost. Finally, we believe that our research points the way toward integrating on-demand crowd work into other authoring interfaces, particularly in creative domains like image editing and programming.

This paper presents Soylent, a word processing interface that uses crowd workers to help with proofreading, document shortening, editing and commenting tasks. Soylent is an example of a new kind of interactive user interface in which the end user has direct access to a crowd of workers for assistance with tasks that require human attention and common sense. Implementing these kinds of interfaces requires new programming patterns for interface software, since crowds behave differently than computer systems. We have introduced one important pattern, Find-Fix-Verify, which splits complex editing tasks into a series of identification, generation, and verification stages that use independent agreement and voting to produce reliable results. We evaluated Soylent with a range of editing tasks, finding and correcting 82% of grammar errors when combined with automatic checking, shortening text to approximately 85% of original length per iteration, and executing a variety of human macros successfully.

Future work falls in three categories. First are new crowd-driven features for word processing, such as readability analysis, smart find-and-replace (so that renaming "Michael" to "Michelle" also changes "he" to "she"), and figure or citation number checking. Second are new techniques for optimizing crowd-programmed algorithms to reduce wait time and cost. Finally, we believe that our research points the way toward integrating on-demand crowd work into other authoring interfaces, particularly in creative domains like image editing and programming.

This paper presents Soylent, a word processing interface that uses crowd workers to help with proofreading, document shortening, editing and commenting tasks. Soylent is an example of a new kind of interactive user interface in which the end user has direct access to a crowd of workers for assistance with tasks that require human attention and common sense. Implementing these kinds of interfaces requires new programming patterns for interface software, since crowds behave differently than computer systems. We have introduced one important pattern, Find-Fix-Verify, which splits complex editing tasks into a series of identification, generation, and verification stages that use independent agreement and voting to produce reliable results. We evaluated Soylent with a range of editing tasks, finding and correcting 82% of grammar errors when combined with automatic checking, shortening text to approximately 85% of original length per iteration, and executing a variety of human macros successfully.

This paper presents Soylent, a word processing interface that uses crowd workers to help with proofreading, document shortening, editing and commenting tasks. Soylent is an example of a new kind of interactive user interface in which the end user has direct access to a crowd of workers for assistance with tasks that require human attention and common sense. Implementing these kinds of interfaces requires new software programming patterns, since crowds behave differently than computer systems. We have introduced one important pattern, Find-Fix-Verify, which splits complex editing tasks into a series of identification, generation, and verification stages that use independent agreement and voting to produce reliable results. We evaluated Soylent with a range of editing tasks, finding and correcting 82% of grammar errors, shortening text to approximately 85% of original length per iteration, and executing a variety of human macros successfully.

This paper presents Soylent, a word processing interface that uses crowd workers to help with proofreading, document shortening, editing and commenting tasks. Soylent is a new kind of user interface in which the end user has direct access to a crowd of workers for assistance with tasks that require human attention and common sense. Implementing such interfaces requires new programming patterns, since crowds behave differently than computer systems. We have introduced one important pattern, Find-Fix-Verify, which splits complex editing tasks into a series of identification, generation, and verification stages to produce reliable results. We evaluated Soylent with a range of editing tasks, finding and correcting 82% of grammar errors, shortening text to approximately 85% of original length per iteration, and executing a variety of human macros successfully.

Figure 1-1: Soylent guides crowd workers to support new kinds of interactions in traditional user interfaces such as the word processor. Here, crowd workers have suggested multiple shorter rewrites of a paragraph, and the user adjusts the length of the paragraph via a slider. Red text indicates locations where rewrites have occurred, in addition to any cuts. Tick marks represent possible lengths, and the blue background bounds the possible lengths.

1.1 Interactive Systems Powered by Crowds

Soylent is a crowd-powered word processor that uses paid micro-contributions to help with writing tasks. Using Soylent is like having an editorial staff available as you write. For example, crowds can shorten the user's writing by finding wordy text and offering alternatives that the user might not have considered. The user selects from these alternatives using a slider that specifies the desired text length (Figure 1-1). Soylent shortens text up to 85% of its original length on average while preserving its meaning. By using tighter wording rather than wholesale cuts, the system shortens papers by lines or pages within minutes. Soylent also offers human-powered proofreading and natural-language macros.

On-demand crowds cost money and can be error-prone; to be worthwhile to the

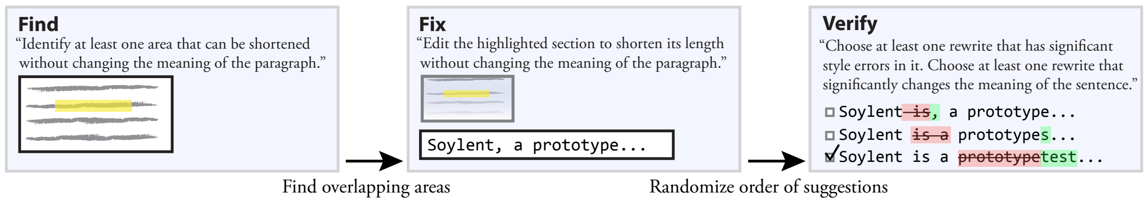

Find
"Identify at least one area that can be shortened without changing the meaning of the paragraph."
Fix
"Edit the highlighted section to shorten its length without changing the meaning of the paragraph."
Soylent, a prototype...
Verify
"Choose at least one rewrite that has significant style errors in it. Choose at least one rewrite that significantly changes the meaning of the sentence."
Soylent ~~is~~, a prototype...
Soylent ~~is a~~ prototypes...
Soylent is a ~~prototype~~test...
Find overlapping areas
Randomize order of suggestions

Figure 1-2: Find-Fix-Verify improves work quality for open-ended tasks such as text shortening by decomposing open-ended tasks into more directed steps.

user, the system must control costs and ensure correctness. To solve this problem, Soylent introduces a crowd programming pattern called *Find-Fix-Verify*. Find-Fix-Verify (Figure 1-2) splits complex tasks into simpler, verifiable steps that utilize independent agreement and voting to produce reliable results. Rather than ask a single crowd worker to read and edit an entire paragraph, for example, Find-Fix-Verify recruits one set of workers to find candidate areas for improvement, then collects a set of candidate improvements, and finally filters out incorrect candidates. This process prevents errant crowd workers from contributing too much or too little, and from introducing errors into the document. This technique guides workers through open-ended editing processes in Soylent. Other researchers have adapted Find-Fix-Verify for object identification in images [152], map labeling [185], and crowd programming languages [145].

1.1.1 Realtime Crowdsourcing Platform and Modeling

Soylent opens a design space of systems that draw on crowd contributions, but existing crowdsourcing platforms are orders of magnitude too slow to provide interactive-speed responses. Many crowd-powered systems need responses in seconds, not minutes or hours. Suppose a developer wanted to build a crowd-powered digital camera. Several-minute response times would be unacceptable, because digital camera users want to see their photos almost immediately.

In response, we introduce *the retainer model* for crowdsourced recruitment, which pays workers a small wage to be on call and respond quickly when asked. This technique cuts wait times for a crowd down from minutes or hours to a median wait time of just two seconds.

To optimize the tradeoff between recruiting too many workers and having too few workers to manage the incoming requests, I develop a mathematical model of retainer recruitment using queueing theory. This model leads to a straightforward maximization algorithm for balancing crowd size against loss rate and cost. A platform running this model can also pursue predictive recruitment by recalling workers in expectation that work will arrive by the time the worker arrives: this technique returns feedback from the crowd just 500 milliseconds after a request.

1.1.2 Realtime Crowd-Powered Systems

Realtime crowds give system designers the ability to recruit crowds for interactive systems within seconds, but work time can still be slow. The next challenge is to guide these crowds quickly as they work, and to demonstrate how realtime crowdsourcing opens opportunities for interactive systems.

Adrenaline is a realtime crowd-powered system: a smart camera shutter powered by crowd intelligence (Figure 1-3). Its goal is to find the right moment to take a photo. Instead of taking a single shot, Adrenaline captures a short video. Video capture allows the user to move around the scene, the subject to strike multiple poses, or action in the scene to unfold unpredictably. Then, Adrenaline identifies the best moment as a still photo about ten seconds later. Low latency means that users can preview and share photos they just took, like they would with any other digital camera.

To deliver realtime results, Adrenaline introduces *rapid refinement*, the first design pattern for synchronous crowds. *Synchronous* crowds are crowds where members arrive and work simultaneously. The retainer model makes it straightforward to recruit synchronous crowds. The fundamental insight behind rapid refinement is that synchronicity enables an algorithm to recognize crowd agreement early. Rapid refinement quickly reduces a large search space by focusing workers' attention on areas where they are beginning to agree independently (Figure 1-4). Repeatedly narrowing the search space to an agreement region increases the quality of the result because it en-

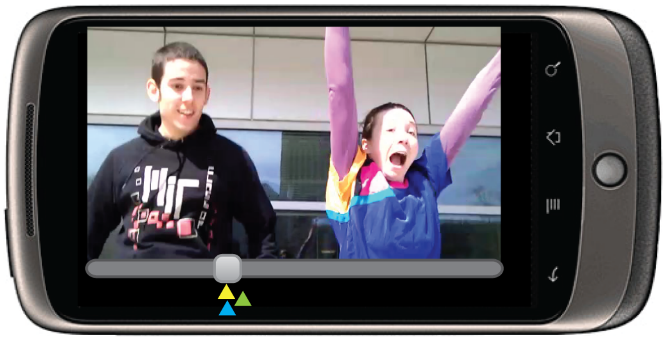


Figure 1-3: Adrenaline's realtime crowds operate a camera shutter by choosing the most photographic moment seconds after it happens. Crowd members' current votes on the best frame appear as colored triangles below the timeline.

forces independent agreement between workers. It also allows the interface to provide incremental, trustable feedback before a final answer is available. Critically, rapid refinement leads to fast results: faster than approaches that keep workers separate, and faster on average than even the fastest individual worker.

1.1.3 Beyond Generic Paid Crowds: Targeted Information Needs

On-demand paid crowds are appropriate for many systems, but a large number of other systems have information needs that generic crowd members may not know. We broaden the scope of crowd-powered systems by demonstrating how social networks and crowd activity traces can support interactive systems where generic paid crowds would struggle.

Friendsourcing

When only a small number of individuals might know the relevant information, social computing design can motivate members of that connected, highly-qualified crowd to contribute. *Friendsourcing* is the use of motivations and incentives over a user's social network to collect information. This technique enables crowd-powered systems that rely on information that generic crowds cannot or do not know. In particular, friendsourcing motivates members of a social network to share accurate information

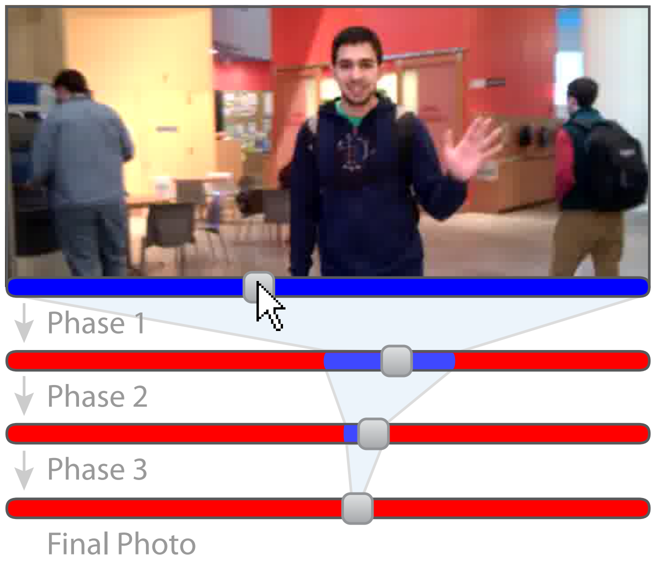


Figure 1-4: Rapid refinement repeatedly shrinks the working area for all workers when it detects that several independent workers are exploring the same area.

about the interests, hobbies, and preferences of people they know. This information can be used to power a personalized system, for example to aid question-answering for topics comprehensible only to a few of a user's friends.

Friendsourcing can either involve creating new kinds of online interactions, or it can facilitate existing information-rich social interactions. *Collabio* is an example of the first class of application, creating new social interactions: it is a social tagging application that has gathered tens of thousands of tags on individuals in a social network. The game collects information that friends know about one another, such as peoples' personalities, expertise, artistic and musical tastes, topics of importance, and even quirky habits. *FeedMe* is an example of the second class of application, facilitating informational social interactions that are already happening. It helps the minority of individuals who consume lots of information on the web share that content with their friends and colleagues, then learns content preferences based on what is shared with each individual.

Mining Crowd Activity Traces

It is not always necessary to motivate or pay crowds to support an interactive system. In many cases, crowds of users have already left activity traces such as browser logs, and this crowd data enables systems to support a large number of less popular user goals. While most system designers focus the design on main user needs, crowd data lets the system adapt dynamically to a vast number of less common needs.

Specifically, search engines can aggregate crowd knowledge to improve broad elements of the search user experience. *Tail Answers* are automatically generated search engine results that support a large set of less common information needs. Examples of uncommon information needs include the normal body temperature for a dog, substitutes for molasses, and the currency in Ireland. Each of these needs may occur thousands of times per year, but are too far in the tail of query traffic to be worth assigning programmers, designers, testers, and product management staff to create and maintain answers. Tail Answers aggregates the knowledge of thousands of everyday web users. Search and browsing patterns suggest web pages where people are finding information to satisfy their original queries, and query logs help identify the specific need. Paid crowds then assist with the final step, extracting the relevant information from the page and promoting it as a direct answer in the search results.

1.2 Contributions

The core contribution of this thesis is the combination of crowdsourcing and computation for interactive systems. Computation helps guide crowds to accomplish tasks that they could not succeed at normally because they struggle with quality or latency. Reciprocally, crowds help interactive systems support a broad new class of applications, tasks, and goals. The sections below synthesize these contributions across multiple systems and applications.

1.2.1 Design of Crowd-Powered Systems

This thesis provides evidence that interactive systems can draw on crowd intelligence to support a wide range of user goals and activities. The systems in this dissertation demonstrate that crowd-powered systems re-open investigation in many areas of human-computer interaction design. For example, Soylent introduces new interactive techniques such as direct-manipulation text shortening, augments existing artificial intelligence systems by supporting copyediting, and supports natural-language commands across text. Adrenaline demonstrates that these concepts work in realtime, for example that a user might take a short video and see the crowd return with the best photographic moment seconds later. These designs give rise to other prototypes that use crowds to generate on-demand designs and vote between alternatives in seconds.

1.2.2 Computational Techniques to Guide Crowds

Crowd-powered systems demonstrate that crowds can participate as first-order elements of interactive software. However, crowds may produce sub-par, high-latency results. This thesis introduces techniques to guide crowds toward higher-quality, faster responses. These techniques are useful across many application contexts, and begin to build a science of software engineering with crowds.

The Find-Fix-Verify design pattern decomposes open-ended tasks such as text editing and shortening into iterative stages. These stages are better-scoped for crowd work, easier to verify, and reduce variance in quality. The pattern is used in Soylent for both proofreading and text shortening.

To produce realtime crowds, the retainer model introduces a new recruitment approach. It hires workers in advance and pays them a small extra wage in exchange for coming back quickly when a task arrives. Empirically, workers return just two seconds after a request is made. We introduce a model of retainer recruitment using queueing theory that allows platforms to directly optimize the tradeoff between crowd size (cost) and expected latency. Retainers currently recruit realtime crowds for Adrenaline, for quick opinion polls, and for crowd-aided design support in photoshop.

Realtime crowds arrive quickly, but they work at a slower pace. So, we introduce rapid refinement, an algorithm that coordinates realtime crowds as they work. Rapid refinement observes early signs of agreement in synchronous crowds and dynamically narrows a continuous search space to focus on promising directions. This approach produces results that, on average, are faster and more reliable than the fastest crowd member working alone.

1.2.3 Generalization to Unpaid Crowds and Specific Needs

Unpaid crowds can extend the reach of these ideas to systems that require specific domain knowledge. This thesis proposes two approaches to engage with non-paid crowd intelligence: designing social computing platforms that motivate domain experts to participate, and mining activity traces from large numbers of users.

Friendsourcing allows systems to target specific populations and create crowds with domain expertise. The technique gathers information using social networks by incentivizing friends and colleagues to share useful information about each other. For example, one friendsourcing application collected over ten thousand tags about friends in a social network. The information in these tags is both accurate, and not available anywhere else on the internet.

By mining activity traces from large numbers of users, systems can recognize and then directly support a large number of less common, long-tail goals. Tail Answers aggregates browsing and searching behavior from millions of sessions in order to find informational queries that are relatively common but have no specific interface designed to respond. It then authors specific, direct results for these queries.

1.3 Thesis Overview

To begin, Chapter 2 provides an introduction to the major research challenges in crowd computing and places this dissertation in the context of related research.

The core of this thesis introduces crowd-powered systems through paid crowdsourcing platforms:

- Chapter 3 introduces the Soylent word processor. Soylent demonstrates how classic interfaces like the word processor can be reinvigorated by including crowds as core elements of the software, for example enabling interactive text shortening, proofreading, and natural language macro commands. Crowds' poor performance on open-ended tasks like document editing motivates the development of the Find-Fix-Verify design pattern, which decomposes open-ended tasks and leads to higher-quality results.

- Chapter 4 focuses on platform support for realtime crowdsourcing. It presents the retainer model for recruiting realtime workers, which can recruit crowds two seconds after a request. It then details a mathematical model of retainers using queueing theory and an optimization algorithm for managing the tradeoff performance guarantees vs. crowd size and cost.

- Chapter 5 uses this platform support to create realtime crowd-powered systems. It introduces Adrenaline, a camera that captures short videos and then coordinates crowds to select the best photographic moment within ten seconds later. Adrenaline's realtime constraints lead to the rapid refinement algorithm, which guides realtime crowds to work quickly by focusing their efforts on areas of emerging agreement.

Chapter 6 moves beyond generic paid crowds to explore how designing social computing platforms and data mining can support systems that paid crowdsourcing would find very difficult. It introduces friendsourcing: the use of motivations and incentives over a user's social network to collect information or produce a desired outcome. It also demonstrates how interactive systems can tap into the large-scale activity traces that crowds leave as they browse the web or use social media.

Finally, the concluding chapters reflect on the opportunities and challenges that this research raises. Chapter 7 puts forth a framework for choosing between different types of intrinsically and extrinsically motivated crowds for an application, the limits of crowdsourcing, and ethical considerations. Chapter 8 reviews the contributions of the dissertation and articulates a vision for the future of crowd computing.

Chapter 2

Related Work

Crowdsourcing is a growing area of research in computer science. Its successes impact areas across the discipline — for instance, artificial intelligence and machine learning, database systems, and human-computer interaction — as well as other disciplines such as management science, economics, and the natural sciences. This chapter lays out the crowdsourcing literature in computer science and related disciplines. Its goal is to lay the groundwork for crowd-powered systems and demonstrate the growth in the area which has followed the systems in this thesis.

2.1 Crowdsourcing

Crowdsourcing is asking a large group of people to help complete a task. It aims to capitalize on the wisdom of crowds [187]: that, under the right conditions, aggregate crowds can be smarter than the single most intelligent person in the group. Crowdsourced tasks can be as large as translating every page on the internet [73] or as small as transcribing a single paragraph of handwriting (Figure 2-1, [130]); as complex as writing a Wikipedia article [104] or as simple as verifying a database entry [201]. The main challenges in any crowdsourcing task are: 1) motivating participation, 2) designing how participants will interact, 3) identifying high-quality answers, and 4) combining individual submissions to complete high-level work.

Major research challenges in crowdsourcing are focused on the border between

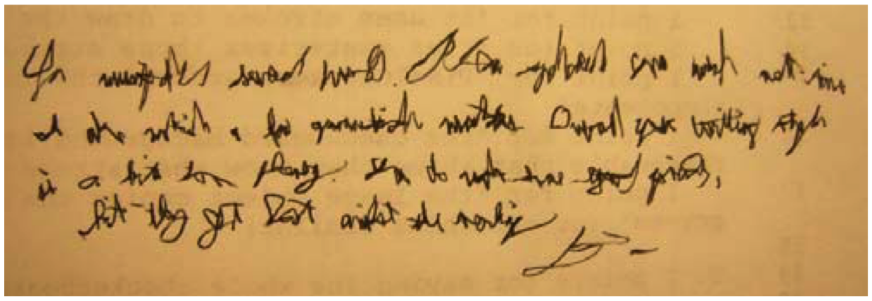

[illegible]

Figure 2-1: Crowd contributions make small improvements to correctly transcribe messy handwriting. While few individuals could transcribe this text correctly, the crowd gets it mostly correct (errors underlined): “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).” The missed words: flowery, get, verbiage, and B-. [130].

computation and human participation. Can algorithms act as management structures and guide crowds through complex work and large-scale tasks [104, 114, 42, 3]? How might crowds and artificial intelligence trade off each others’ strengths [94]? What kinds of volunteer, social, monetary, or game incentives will drive participation [199, 198, 141, 37, 109]? How can systems separate high-quality work from low-quality work without being sure of the correct answer on most tasks [124, 174, 203]? Finally, the question that this thesis poses: what are the opportunities and techniques for building deployable interactive software with crowds inside?

2.1.1 Definitions and Design Space

Jeff Howe introduced the term *crowdsourcing* to draw comparisons to *outsourcing*, or pushing work beyond a company’s walls [85, 86]. Where outsourcing pays a few external contractors, crowdsourcing makes an open call to the public for help[1]. Simultaneously, the academic literature has given rise to related concepts such as *human computation* [199], *collective intelligence* [132], and *social computing* [158]. While each of these concepts has meaningful distinctions from crowdsourcing — human compu-

[1]Admittedly, paid crowdsourcing platforms such as Amazon Mechanical Turk have since muddled this distinction.

tation sees humans as computational elements [122], social computing concerns itself with large-scale human phenomena online, and collective intelligence restricts itself to intelligent behavior — I will use the term crowdsourcing for much of this thesis[2]. When I want to stress that computation and algorithms are guiding the crowd, I will use the term *crowd computing*.

Law and von Ahn [122], Malone et al. [132], and Quinn and Bederson [161] offer overviews of the crowdsourcing literature. Quinn frames the design space as decisions over motivation, quality control, aggregation policy, expertise, computer-human order, and task-request cardinality. Law and von Ahn add that task routing, algorithm, and task design also play an important role. Malone simplifies the design space into questions of 1) *What?*, 2) *Why?*, 3) *How?*, and 4) *Who?* Techniques in this thesis such as Find-Fix-Verify cross several dimensions: they aim to improve quality control through algorithmic processes and task design. However, each dimension individually receives attention in the research literature. For example, quality control processes and spam detection are common in the literature: some example techniques involve estimating worker quality jointly with the answer [174], using gold-standard questions [124], and providing multiple parallel task pathways [68].

In *When Computers Were Human* [67], David Alan Grier makes the case that distributed human computation reaches at least as far back as the 18th century. The British Astronomer Royal began recruiting colleagues to help perform calculations for nautical charts and traded calculation tables through the mail. This original instantiation of human computation reached its height in the 1930's with a Works Progress Administration organization called the Mathematical Tables Project. The project primarily created large almanacs for the values of mathematical functions, but also aided in the computation of ballistics tables. The Mathematical Tables Project hired roughly 450 so-called *human computers* without significant mathematical expertise, then simplified and structured the process to get accurate results. Human computers are actually the source of the term *computer* that we use today: electronics took over

[2]Quinn suggests one organizing set of definitions for crowdsourcing, human computation, and social computing [161].

these responsibilities from the original computers.

2.1.2 Voluntary Crowdsourcing

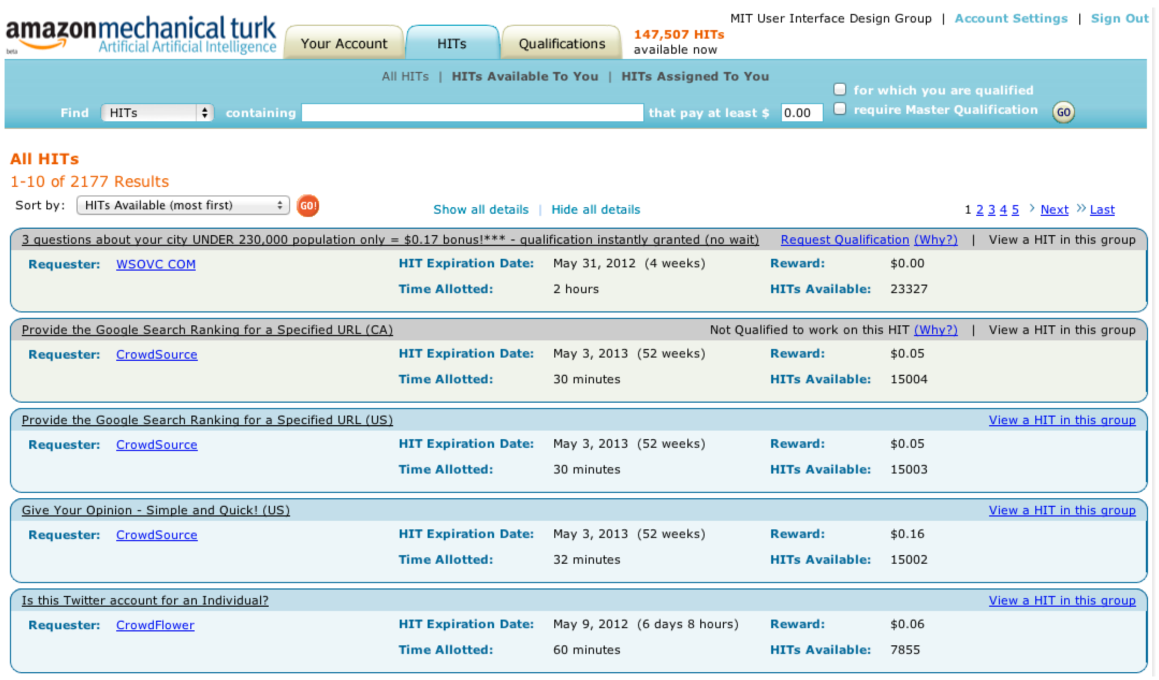
Many popular crowdsourcing platforms depend on volunteerism or other non-monetary incentives for participation. For example, volunteer crowds have:

- Authored Wikipedia[3], the largest encyclopedia in history,
- Helped NASA identify craters on the moon [95],
- Surveyed satellite photos for images of a missing person [81],
- Held their own in chess against a world champion [148],
- Solved open mathematics problems [38],
- Generated large datasets for object recognition [168, 199],
- Collected eyewitness reports during crises and violent government crackdowns [153], and
- Generated a large database of common-sense information [179].

Each of these successes relied on the individuals' intrinsic motivation to participate in the task.

Beginning with Luis von Ahn's ESP Game [199], games have proven themselves to be a powerful mechanism to attract crowds. Games with a Purpose [198] require that the designer turn a task that is useful to computers but typically boring, such as labeling images [199], rating photos [72], or building 3D models [193], into one that is completed as the side effect of a game. By designing the game correctly, researchers can support scientific discovery. For example, FoldIt discovered the structure of proteins that stumped scientists for fifteen years [37].

[3]`http://www.wikipedia.org`


amazonmechanical turk
Artificial Artificial Intelligence
Your Account
HITs
Qualifications
147,507 HITs available now
MIT User Interface Design Group | Account Settings | Sign Out
All HITs | HITs Available To You | HITs Assigned To You
Find HITs containing that pay at least $ 0.00
for which you are qualified
require Master Qualification
GO
All HITs
1-10 of 2177 Results
Sort by: HITs Available (most first)
GO!
Show all details | Hide all details
1 2 3 4 5 Next Last

3 questions about your city UNDER 230,000 population only = $0.17 bonus!*** - qualification instantly granted (no wait) — Request Qualification (Why?) | View a HIT in this group

Requester:	WSOVC COM	HIT Expiration Date:	May 31, 2012 (4 weeks)	Reward:	$0.00
		Time Allotted:	2 hours	HITs Available:	23327

Provide the Google Search Ranking for a Specified URL (CA) — Not Qualified to work on this HIT (Why?) | View a HIT in this group

Requester:	CrowdSource	HIT Expiration Date:	May 3, 2013 (52 weeks)	Reward:	$0.05
		Time Allotted:	30 minutes	HITs Available:	15004

Provide the Google Search Ranking for a Specified URL (US) — View a HIT in this group

Requester:	CrowdSource	HIT Expiration Date:	May 3, 2013 (52 weeks)	Reward:	$0.05
		Time Allotted:	30 minutes	HITs Available:	15003

Give Your Opinion - Simple and Quick! (US) — View a HIT in this group

Requester:	CrowdSource	HIT Expiration Date:	May 3, 2013 (52 weeks)	Reward:	$0.16
		Time Allotted:	32 minutes	HITs Available:	15002

Is this Twitter account for an Individual? — View a HIT in this group

Requester:	CrowdFlower	HIT Expiration Date:	May 9, 2012 (6 days 8 hours)	Reward:	$0.06
		Time Allotted:	60 minutes	HITs Available:	7855

Figure 2-2: Amazon Mechanical Turk presents a list of tasks and payment for each. The list can be sorted by price, number of tasks, and recency. Workers accept a task, complete it, and submit it to the requester for review.

Other non-monetary incentives are possible as well. Duolingo[4] aims to help people learn — in particular learn a language — and translate the web into many languages as a side effect. Crowds will complete tasks as part of a web security measures such as CAPTCHAs [200], or share their expertise in response to rewards tuned to the local community [80].

2.1.3 Paid Crowdsourcing

While voluntary [179] and game-based participation [198, 37] spawned academic interest in crowdsourcing, paid microwork platforms have helped broaden it. Paid crowdsourcing markets allow requesters to post tasks and offer monetary incentives. By solving the motivation question through extrinsic incentives, these platforms allow researchers to study how crowds act without needing to design games or tap into volunteerism.

[4]`http://www.duolingo.com`

Paid crowdsourcing markets typically present a list of tasks and monetary rewards for each task (Figure 2-3). Workers browse these lists to choose a task. Depending on the platform, workers will: 1) complete the task immediately and submit it for review, or 2) bid on the task and wait for the requester to choose them for the work. Again depending on the platform, requesters may pay everyone who participates, reject unsatisfactory work, and/or only pay for the best submission.

Perhaps the most well-known paid crowdsourcing platform is Amazon Mechanical Turk[5]. Mechanical Turk workers complete over five million tasks each year, typically in a range of one to ten cents each [89]. Amazon borrowed the name "Mechanical Turk" from a fake chess automaton built during the 18th century. The machine appeared to be an extremely skilled chess player, but it was a ruse: a human hid inside the machine and played the game from within. Likewise, Amazon stylizes the platform as "artificial artificial intelligence": appropriate for tasks where AI could not yet succeed.

Many choices can affect worker behavior, including price, time of day, worker demographics, and training. Higher wages result in work being completed more quickly but not at higher quality [141]. More specifically, the half-life for tasks ranges between twelve hours and two days depending on the wage [202]. Like many online phenomena, distributions for worker tasks completed, requester tasks posted, and completion times follow a roughly lognormal distribution [89]. Workers typically find tasks on Mechanical Turk by sorting the task listing either by recency (trying to find new opportunities) or number of tasks available (trying to complete a number of tasks at once) [32]. Peer or expert review of tasks can improve work quality [48].

Workers on Mechanical Turk are roughly 40% from the United States, 40% from India, and 20% from elsewhere in the world [167]. Across indices such as gender, education and income, the workers reflect the overall population distributions for these countries [167]. So, there are educated individuals on these platforms who work to supplement or entirely replace their other income streams using Mechanical Turk. Workers in India are more motivated by monetary incentives than those in the United

[5]`http://www.mturk.com`

States [6].

There are paid crowdsourcing systems other than Amazon Mechanical Turk. Prediction markets allow participants to wager on outcomes such as election winners, then make money if they are correct [133]. A large team of distributed participants found red balloons that DARPA had located across the country and shared the reward money [189]. Sites such as Freelancer[6] and oDesk[7] offer a similar platform to Mechanical Turk, except oriented toward expert work and higher pay. MobileWorks [149] and mClerk [71] use paid crowdsourcing to help lift workers in developing regions out of poverty.

Paid crowdsourcing raises questions about labor issues. Ensuring that workers can make a living wage is one important concern [54]. Bederson and Quinn suggest that the balance may be tipped too far toward requesters currently, leaving workers with less ability to raise concerns or appeal decisions [10]. In response, workers develop forums and applications that allow them to organize and warn each other away from bad requesters [176, 175]. I will return to this discussion and propose next steps in Chapter 7.

Uses of Paid Crowdsourcing

Paid crowds have proven quite useful for gathering labeled data, used later to train machine learning algorithms. For example, crowds can match expert annotation abilities on natural language processing tasks such as affect and word sense disambiguation [181, 28]. Likewise, paid crowds generate large speech corpuses for spoken language research [28, 142]. Visual domains are also obvious choices: in machine vision, paid crowds annotate objects and people in images [183], and in graphics, they help with tasks such as identifying depth layers [62]. Even when not training algorithms, crowds can solve algorithmic problems: for example, groups of individuals with limited communication abilities can still solve NP-complete problems such as graph-coloring [97, 134].

[6]`http://www.freelancer.com/`
[7]`http://www.odesk.com`

Figure 2-3: The Sheep Market is an art piece exploring crowd-authored creativity. This image shows a small subset of the 10,000 sheep drawn by workers on Mechanical Turk [107].

While Mechanical Turk is used for many difficult artificial intelligence problems, it also is a platform to study large-scale human interactions and is a ready subject pool for behavioral experiments. By structuring tasks so that it is difficult to shirk, researchers can use Mechanical Turk for user studies [102]. Mason and Suri generalized these ideas into a set of recommendations for behavioral research on Mechanical Turk [140]. With qualification tests, crowds are also appropriate for perception and visualization experiments [79]. Finally, by posting different versions of an interface at multiple price points and tracking how quickly the tasks are completed, designers can quantify the cost (utility) of design decisions [191].

Beyond traditional forced-choice tasks, crowds can also undertake generative, creative challenges. *The Sheep Market* was one of the first such pieces, creating a large tapestry out of individually-drawn sheep [107]. Likewise, Mechanical Turk workers have written short stories and illustrated them [76]. Genetic algorithms can coordinate the creative process to cross-pollinate ideas as well [206].

2.2 Crowd-Powered Systems

While the systems in this thesis helped establish crowd-powered systems as a research field, there is a growing literature in the area that informed and builds on this work.

Human computation has been a key tool in interactive systems since the development of Wizard of Oz studies in the 1980's [99]. Wizard of Oz studies are a standard prototyping approach in human-computer interaction. In a Wizard of Oz study, a human simulates the intelligence that will eventually be programmed into the system. The point is that designers can gather feedback on whether the system will be useful before they put in time engineering the system. This dissertation suggests that it may be possible to transition from an era where Wizard of Oz techniques were used only as prototyping tools to an era where a "Wizard of Turk" can be permanently wired into a system.

One of the first computing systems to integrate paid crowds was PEST [170]. PEST used Mechanical Turk to vet advertisement recommendations. CrowdSearch used crowds to help perform image searches on mobile phones [205], and ChaCha[8] used a private crowd to answer users' questions when mobile. However, these systems all consist of a single user operation and little or no interaction. This dissertation extends this work to more creative, complex tasks where the user can make personalized requests and interact with the returned data by direct manipulation.

VizWiz, another crowd-powered system, allows blind users to ask questions about their surroundings [21]. The system also introduces quikTurKit, which recruits workers in advance to solve old tasks, then replaces the old tasks with new work when requests arrive. It also re-posts tasks to keep them near the top of the "most recent" ordering, which is a popular view that workers use to find tasks [32]. This thesis adopts versions of both these techniques, then introduces new techniques to gather all crowd members simultaneously, reduce variance in lag times, and reduce wait times by a factor of ten — without added cost.

In the eighteen months since the introduction of Soylent, the research literature has articulated a number of additional crowd-powered systems. For example, in the health domain, crowds can help track calorie intake [152] and help users respond to stress positively [147]. Robots can call on crowd help when the robot is uncertain about actions to take [182], or crowds can collaboratively take complete control of a robot [117]. Task and todo planning is another viable area: Mobi helps users tackle task planning with global constraints, for example preparing for a vacation

[8]`http://www.chacha.com/`

[207]. Finally, crowds can support authoring and prototyping tools: CrowdSight helps developers prototype applications that integrate machine vision [165], and CollabMap helps author maps [185].

The database community has produced systems and techniques that demonstrate the power of integrating crowd intelligence into data management systems. These systems can call on the crowd to dynamically create rows or tables [56] — effectively producing an open-world database — and can be queried using SQL-style languages [138, 157]. This formalization means that it is natural to consider optimization approaches for crowdsourced data, for example sorts and joins [137], filters [156], counting [136], and finding the item which maximizes some value [70].

2.2.1 Soylent

The crowd-powered systems in this thesis each draw on related system designs and artificial intelligence algorithms.

Soylent is inspired by writers' reliance on friends and colleagues to help shape and polish their writing [47]. But we cannot always rely on colleagues: they do not want to proofread every sentence we write, cut a few lines from every paragraph in a ten-page paper, or help us format thirty ACM-style references.

Proofreading is emerging as a common task on Mechanical Turk. Standard Minds[9] offers a proofreading service backed by Mechanical Turk that accepts plain text via a web form and returns edits one day later. By contrast, Soylent is embedded in a word processor, has much lower latency, and presents the edits in Microsoft Word's user interface. Our work also contributes the Find-Fix-Verify pattern to improve the quality of such proofreading services.

Automatic proofreading has a long history of research [112] and has seen successful deployment in word processors. However, Microsoft Word's spell checker frequently suffers from false positives, particularly with proper nouns and unusual names. Its grammar checker suffers from the opposite problem: it misses blatant errors[10]. Hu-

[9]`http://standardminds.com/`
[10]`http://faculty.washington.edu/sandeep/check`

man checkers are currently more reliable, and can also offer suggestions on how to fix the errors they find, which is not always possible for Word — for example, consider the common (but mostly useless) Microsoft Word feedback, “Fragment; consider revising.”

Soylent’s text shortening component is related to document summarization, which has also received substantial research attention [135]. Microsoft Word has a summarization feature that uses sentence extraction, which identifies whole sentences to preserve in a passage and deletes the rest, producing substantial shortening but at a great cost in content. Shortn’s approach, which can rewrite or cut parts of sentences, is an example of sentence compression, an area of active recent research [36, 106] that suffers from a lack of training data [34]. Soylent’s results produce training data to help push this research area forward.

The Human Macro is related to AI techniques for end-user programming. Several systems allow users to demonstrate repetitive editing tasks for automatic execution; examples include Eager, TELS, and Cima [41], LAPIS [143], and SmartEdit [126]. Other work has considered programming syntax similar to natural language [131].

2.2.2 Adrenaline and Realtime Crowd-Powered Systems

Adrenaline shares the goals of the Moment Camera [35] by recording continuously and choosing the best moment for a photo. The Moment Camera and Adrenaline complement each other. Computational photography can check whether subjects’ eyes are open and can tune camera settings, but it is only trained on certain classes of images and it is much harder to train an algorithm to make subjective judgments. This thesis will suggest that crowds are effective at making the hard semantic decision of what constitutes a good photo.

Adrenaline relies on synchronous crowds collaborating implicitly. For example, paid crowds can solve distributed problems like graph coloring [134]. Collaborative text authoring software enables paid workers to pursue collaborative text translation [101]. The Shepherd system likewise recognizes that workers overlapping in time could provide synchronous peer feedback [48]. The retainer model can bring crowds of this kind together relatively quickly, further enabling this kind of work. Many of these techniques draw on synchronous group collaboration research (e.g., [66, 91]).

2.3 Crowds and Algorithms

Computation and algorithms can coordinate crowds to accomplish tasks by structuring the crowd's work process. While most crowdsourcing tasks are single calls to action — for example, gathering labels on an image — algorithms can coordinate entire workflows. The Improve-and-Vote algorithm asks crowd members to iteratively edit the previous worker's submission and vote on the best improvement [128]. Improve-and-Vote can accomplish tasks such as transcribing incredibly messy handwriting (Figure 2-1). This thesis contributes Find-Fix-Verify, a design pattern for decomposing tasks which are much more open-ended than common crowdsourcing goals, and Rapid Refinement, an algorithm for guiding crowds through a large search space quickly. By generalizing Find-Fix-Verify's decomposition approach, it is possible to create a MapReduce-style framework that splits sub-tasks and recombines work [104], as well as guide crowds to author their own workflows by recursively splitting tasks that are too large [114].

Writing crowd algorithms requires improved toolkits, APIs and languages. TurKit [129] proposes an iterative programming approach that integrates with common languages (e.g., `mturk.ask("What color shirt are you wearing?")`. The program blocks while the user waits for a response, and saves old worker responses so that the request returns immediately the next time the program is run or debugged. Dog [3], Qurk [138, 137], and Deco [157] propose declarative languages similar to SQL and Pig [154] for data-centric tasks. Formal programming languages for crowds can also help the programmer understand control errors that the crowd might make [9]: these languages use techniques such as Find-Fix-Verify as benchmark tasks [145].

Artificial intelligence aims to optimize and guide these workflows. For example, TurKontrol models the Improve-and-Vote algorithm using decision theory tools (POMDPs) and only recruits as many workers as necessary for each iteration and vote stage based on the algorithm's current level of uncertainty [42, 43]. Shifting more towards an active learning framework, algorithms can also ask crowds to label the most uncertain dimensions of the input [24]. Likewise, algorithms can jointly optimize processes that combine crowd and machine learning contributions [94, 134]. Rather than iterative improvements, genetic algorithms can also structure and cross-pollinate the information passed between crowd workers in each phase [108, 206]. Each of the algorithms and workflows in this thesis could be further optimized by applying these techniques.

Finally, it is worth noting that spam and quality control continue to be an issue on crowdsourcing platforms. The typical goal of research in this space is to jointly learn the correct answers and worker quality. One approach is to treat this problem analogously to an Expectation Maximization algorithm: using the worker responses to guess answers, then the guessed answers to improve weights on worker responses [45, 44]. Then, by additionally modeling worker bias, the system can recommend whether it would be useful to ask for an additional worker vote [174, 88]. Alternatively, the system can observe implicit signals of worker quality — number of keystrokes, scrolling behavior, time to completion — and train models that predict worker quality without looking at the response at all [169].

2.4 Social Network-based Crowds

Often, people make requests not to generic crowds but to targeted groups of friends and colleagues. For example, people often ask their own social networks for information or advice in a phenomenon known as *social search* [51, 155]. This practice draws on years of research on organizational knowledge-sharing [1]. Systems can support and facilitate social search by routing questions or making the funcitonality more readily visible [146, 23, 84]. This thesis introduces *friendsourcing*, which generalizes this phenomenon so that social network crowds might solve problems and collect data beyond question-answering.

Social networks are well-positioned to share accurate and novel information. For example, a combination of external and self-rated responses to personality surveys produces a more accurate picture than either the external raters or the individual alone [195], suggesting that the integration of friends' impressions into profiles may lead to more accurate portrayals. Furthermore, a large number of members of a social network do not actively contribute information — 41% of profile fields on Facebook are missing [115]. Friendsourcing is able to gather information about a large number of individuals who are less active.

However, studies of contribution in online communities suggest several potential challenges with social network crowds. One danger is *social loafing*: users will exhibit little effort on a collective task when they believe that others will also contribute [96, 121]. Related to social loafing is *diffusion of responsibility*: when many individuals share the responsibility for an action that one person must perform, each feels less cognitive dissonance when he or she shirks [120]. Social loafing and the diffusion of responsibility together lead to the phenomenon known as the *bystander effect*. The bystander effect exists in computer-mediated communication, for example in chat rooms where a newcomer asks a full chatroom for technical help but nobody steps forward to answer the question [139].

Previous work has found that individuals are likely to contribute to an online community when they are reminded of the uniqueness of their contributions, are given specific, challenging goals, and are helping groups similar to themselves [162, 11, 109]. Thus, in friendsourcing, we challenge individuals' (potentially obscure) knowledge of members of their own social group. Both active and loafing users can be motivated by comparing their activity to the median participation of the community [74], for example via leaderboards in Collabio and FeedMe. Loafing can also be overcome via opportunities for reciprocity toward other friends [163], motivating Collabio's Facebook notifications upon tagging a friend.

2.5 Mining Crowd Data

Several systems power novel interactions by mining the wisdom of crowds. HelpMeOut [77] collects debugging traces and applies others' error solutions to help fix code. MySong [178] indexes a library of music chords to enable the user to build a chord progression by singing a melody line. Google Suggest mines query logs to speed and direct new queries. Sketch2Photo [31] transforms a hand-sketched photo outline annotated with descriptive terms and produces a composite photo by searching a large database of images.

On-demand crowds are useful in that they expand the realm of tasks we can support beyond those requiring traces or incentives. However, activity traces may carry more information about unusual needs and goals.

Tail Answers in particular extends work on automatic answers in information retrieval. Many question-answering systems are designed to address information needs with short phrases such as using search result n-grams to identify answers [127, 25, 2]. A second approach is open-domain information extraction, for example TextRunner [8]. These approaches work best when facts are repeated across multiple web pages. Finally, systems can employ curated knowledge bases such as YAGO [186] and match on them to answer some queries. However, automated approaches can make mistakes that are obvious to humans.

2.6 Conclusion

Computing is helping crowds solve problems that were previously out of reach. Simultaneously, crowds are aiding computation by gathering training data, evaluating systems, and providing on-demand cognition. This thesis synthesizes these ideas, drawing on crowd intelligence to support interactive computing and drawing on computing to help guide crowds.

Chapter 3

Soylent: A Word Processor with a Crowd Inside

This thesis suggests that crowds can reshape the character of interactive systems to be more natural and powerful. To do this, I will take up the word processor as a classic interactive system and demonstrate that crowds, guided by computation, re-open core questions of the design and implementation of such systems.[1]

Word processors may well be the most heavily-designed, heavily-used interactive systems ever. They support a deep cognitive activity — writing — and support complicated manual manipulations. Word processors have traditionally focused on offloading manual tasks, for example layout [116], spell checking and grammar checking [112].

However, word processors still fall far short of supporting that cognitive process they were originally designed to facilitate: writing. They do not currently help with core tasks such as expressivity, word choice, ideas, structure, organization, or references. Even the existing support is imperfect: conscientious users still routinely leave style, grammar and spelling mistakes.

In our everyday life, when we need help with complex cognition and manipulation tasks, we often turn to other people. We ask friends to answer questions that we

[1]This chapter has adapted, updated, and rewritten content from a paper at UIST 2010 [18].

cannot answer ourselves [51]; masses of volunteer editors flag spam edits on Wikipedia [110]. Writing is no exception [47]: we commonly recruit friends and colleagues to help us shape and polish our writing. But we cannot always rely on them: colleagues do not want to proofread every sentence we write, cut a few lines from every paragraph in a ten-page paper, or help us format thirty ACM-style references.

Soylent is a word processing interface that utilizes crowd contributions to aid complex writing tasks ranging from error prevention and paragraph shortening to automation of tasks such as citation searches and tense changes. Using Soylent is like having an entire editorial staff available as you write. We hypothesize that crowd workers with a basic knowledge of written English can support both novice and expert writers. These workers perform tasks that the writer might not, such as scrupulously scanning for text to cut or updating a list of addresses to include a zip code. They can also solve problems that artificial intelligence cannot yet, for example flagging writing errors that the word processor does not catch.

Soylent aids the writing process by integrating paid crowd workers from Amazon's Mechanical Turk platform into Microsoft Word. *Soylent is people:* its core algorithms involve calls to Mechanical Turk workers (Turkers). Soylent is comprised of three main components:

1. *Shortn*, a text shortening service that cuts selected text down to 85% of its original length on average without changing the meaning of the text or introducing writing errors.

2. *Crowdproof*, a human-powered spelling and grammar checker that finds problems Word misses, explains the error, and suggests fixes.

3. *The Human Macro*, an interface for offloading arbitrary word processing tasks such as formatting citations or finding appropriate figures.

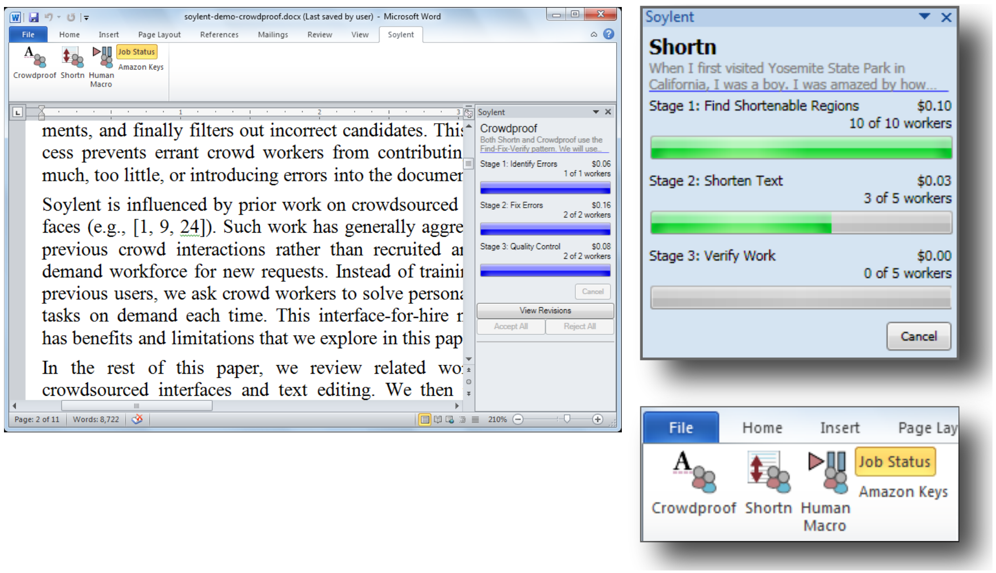
The main contribution of Soylent is *the idea of embedding paid crowd workers in an interactive user interface to support complex cognition and manipulation tasks on demand*. These crowd workers do tasks that computers cannot reliably do automatically and the user cannot easily script. This chapter contributes the design of one such system, an implementation embedded in Microsoft Word, and a programming pattern that increases the reliability of paid crowd workers on complex tasks. It then expands these contributions with feasibility studies of the performance, cost, and time delay of our three main components and a discussion of the limitations of our approach with respect to privacy, delay, cost, and domain knowledge.

The fundamental technical contribution of this system is a crowd programming pattern called *Find-Fix-Verify*. Mechanical Turk costs money and it can be error-prone; to be worthwhile to the user, we must control costs and ensure correctness. Find-Fix-Verify splits complex crowd intelligence tasks into a series of generation and review stages that utilize independent agreement and voting to produce reliable results. Rather than ask a single crowd worker to read and edit an entire paragraph, for example, Find-Fix-Verify recruits one set of workers to find candidate areas for improvement, another set to suggest improvements to those candidates, and a final set to filter incorrect candidates. This process prevents errant crowd workers from contributing too much or too little, or introducing errors into the document.

In the rest of this chapter, we introduce Soylent and its main components: Shortn, Crowdproof, and The Human Macro. We detail the Find-Fix-Verify pattern that enables Soylent, then evaluate the feasibility of Find-Fix-Verify and our three components.

3.1 Soylent

Soylent is a prototype crowdsourced word processing interface. It is currently built into Microsoft Word (Figure 3-1), a popular word processor and productivity application. It demonstrates that computing systems can reach out to crowds to: 1) create new kinds of interactive support for text editing, 2) extend artificial intelligence systems such as style checking, and 3) support natural language commands. These three goals are embedded in Soylent's three main features: text shortening, proofreading, and arbitrary macro tasks.



ments, and finally filters out incorrect candidates. Thi
cess prevents errant crowd workers from contributin
much, too little, or introducing errors into the documer

Soylent is influenced by prior work on crowdsourced
faces (e.g., [1, 9, 24]). Such work has generally aggre
previous crowd interactions rather than recruited a
demand workforce for new requests. Instead of traini
previous users, we ask crowd workers to solve person
tasks on demand each time. This interface-for-hire r
has benefits and limitations that we explore in this pap

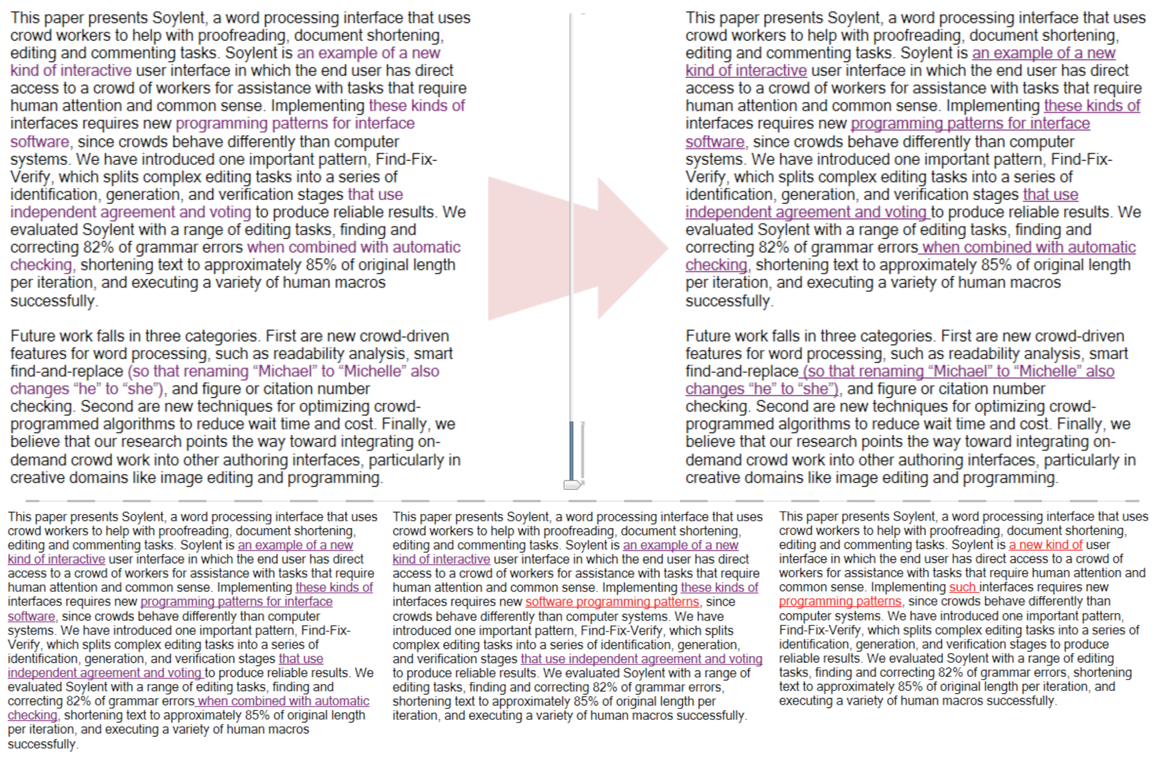
In the rest of this paper, we review related wo
crowdsourced interfaces and text editing. We then

Figure 3-1: Soylent adds a set of commands and a status bar to Microsoft Word. We envision that these same concepts could be translated to many similar systems.

3.1.1 Shortn: Text Shortening

Shortn aims to demonstrate that crowds can support new kinds of interactions and interactive systems that were very difficult to create before. Some authors struggle to remain within length limits on papers and spend the last hours of the writing process tweaking paragraphs to shave a few lines. This is painful work and a questionable use of the authors' time. Other writers write overly wordy prose and need help editing. Automatic summarization algorithms can identify relevant subsets of text to cut [135]. However, these techniques are less well-suited to small, local language tweaks like those in Shortn, and they cannot guarantee that the resulting text flows well.

Soylent's Shortn interface allows authors to condense sections of text. The user selects the area of text that is too long — for example a paragraph or section — then presses the Shortn button in Word's Soylent command tab (Figure 3-1). In response, Soylent launches a series of Mechanical Turk tasks in the background and notifies the user when the text is ready. The user can then launch the Shortn dialog box (Figure 3-2). On the left is the original paragraph; on the right is the proposed revision. Shortn provides a single slider to allow the user to continuously adjust the length



This paper presents Soylent, a word processing interface that uses crowd workers to help with proofreading, document shortening, editing and commenting tasks. Soylent is an example of a new kind of interactive user interface in which the end user has direct access to a crowd of workers for assistance with tasks that require human attention and common sense. Implementing these kinds of interfaces requires new programming patterns for interface software, since crowds behave differently than computer systems. We have introduced one important pattern, Find-Fix-Verify, which splits complex editing tasks into a series of identification, generation, and verification stages that use independent agreement and voting to produce reliable results. We evaluated Soylent with a range of editing tasks, finding and correcting 82% of grammar errors when combined with automatic checking, shortening text to approximately 85% of original length per iteration, and executing a variety of human macros successfully.

Future work falls in three categories. First are new crowd-driven features for word processing, such as readability analysis, smart find-and-replace (so that renaming "Michael" to "Michelle" also changes "he" to "she"), and figure or citation number checking. Second are new techniques for optimizing crowd-programmed algorithms to reduce wait time and cost. Finally, we believe that our research points the way toward integrating on-demand crowd work into other authoring interfaces, particularly in creative domains like image editing and programming.

This paper presents Soylent, a word processing interface that uses crowd workers to help with proofreading, document shortening, editing and commenting tasks. Soylent is an example of a new kind of interactive user interface in which the end user has direct access to a crowd of workers for assistance with tasks that require human attention and common sense. Implementing these kinds of interfaces requires new programming patterns for interface software, since crowds behave differently than computer systems. We have introduced one important pattern, Find-Fix-Verify, which splits complex editing tasks into a series of identification, generation, and verification stages that use independent agreement and voting to produce reliable results. We evaluated Soylent with a range of editing tasks, finding and correcting 82% of grammar errors when combined with automatic checking, shortening text to approximately 85% of original length per iteration, and executing a variety of human macros successfully.

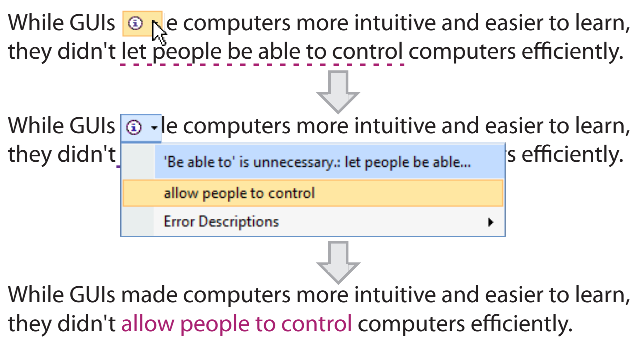
Future work falls in three categories. First are new crowd-driven features for word processing, such as readability analysis, smart find-and-replace (so that renaming "Michael" to "Michelle" also changes "he" to "she"), and figure or citation number checking. Second are new techniques for optimizing crowd-programmed algorithms to reduce wait time and cost. Finally, we believe that our research points the way toward integrating on-demand crowd work into other authoring interfaces, particularly in creative domains like image editing and programming.

This paper presents Soylent, a word processing interface that uses crowd workers to help with proofreading, document shortening, editing and commenting tasks. Soylent is an example of a new kind of interactive user interface in which the end user has direct access to a crowd of workers for assistance with tasks that require human attention and common sense. Implementing these kinds of interfaces requires new programming patterns for interface software, since crowds behave differently than computer systems. We have introduced one important pattern, Find-Fix-Verify, which splits complex editing tasks into a series of identification, generation, and verification stages that use independent agreement and voting to produce reliable results. We evaluated Soylent with a range of editing tasks, finding and correcting 82% of grammar errors when combined with automatic checking, shortening text to approximately 85% of original length per iteration, and executing a variety of human macros successfully.

This paper presents Soylent, a word processing interface that uses crowd workers to help with proofreading, document shortening, editing and commenting tasks. Soylent is an example of a new kind of interactive user interface in which the end user has direct access to a crowd of workers for assistance with tasks that require human attention and common sense. Implementing these kinds of interfaces requires new software programming patterns, since crowds behave differently than computer systems. We have introduced one important pattern, Find-Fix-Verify, which splits complex editing tasks into a series of identification, generation, and verification stages that use independent agreement and voting to produce reliable results. We evaluated Soylent with a range of editing tasks, finding and correcting 82% of grammar errors, shortening text to approximately 85% of original length per iteration, and executing a variety of human macros successfully.

This paper presents Soylent, a word processing interface that uses crowd workers to help with proofreading, document shortening, editing and commenting tasks. Soylent is a new kind of user interface in which the end user has direct access to a crowd of workers for assistance with tasks that require human attention and common sense. Implementing such interfaces requires new programming patterns, since crowds behave differently than computer systems. We have introduced one important pattern, Find-Fix-Verify, which splits complex editing tasks into a series of identification, generation, and verification stages to produce reliable results. We evaluated Soylent with a range of editing tasks, finding and correcting 82% of grammar errors, shortening text to approximately 85% of original length per iteration, and executing a variety of human macros successfully.

Figure 3-2: Shortn allows users to adjust the length of a paragraph via a slider. Red text indicates locations where rewrites have occurred, in addition to any cuts. Tick marks represent possible lengths, and the blue background bounds the possible lengths.


While GUIs ⓘ [cursor]e computers more intuitive and easier to learn, they didn't let people be able to control computers efficiently.

While GUIs ⓘ ▾le computers more intuitive and easier to learn, they didn't … s efficiently.

While GUIs made computers more intuitive and easier to learn, they didn't allow people to control computers efficiently.

Figure 3-3: Crowdproof is a human-augmented proofreader. The drop-down explains the problem (blue title) and suggests fixes (gold selection).

of the paragraph. As the user does so, Shortn computes the combination of crowd trimmings that most closely match the desired length and presents that text to the user on the right. From the user's point of view, as she moves the slider to make the paragraph shorter, sentences are slightly edited, combined and cut completely to match the length requirement. Areas of text that have been edited or removed are highlighted in red in the visualization. These areas may differ from one slider position to the next.

Shortn typically can remove up to 15–30% of a paragraph in a single pass, and up to 50% with multiple iterations. It preserves meaning when possible by encouraging workers to focus on wordiness and separately verifying that the rewrite does not change the user's intended meaning. Removing whole arguments or sections is left to the user.

3.1.2 Crowdproof: Crowdsourced Copyediting

Shortn demonstrates that crowds can power new kinds of interactions. We can also involve crowds to augment the artificial intelligence built into applications, for example proofreading. Crowdproof instantiates this idea.

Soylent provides a human-aided spelling, grammar and style checking interface called Crowdproof (Figure 3-3). The process finds errors, explains the problem, and offers one to five alternative rewrites. Crowdproof is essentially a distributed proofreader or copyeditor.

To use Crowdproof, the user highlights a section of text and presses the proofreading button in the Soylent ribbon tab. The task is queued to the Soylent status pane and the user is free to keep working. Because Crowdproof costs money, it does not issue requests unless commanded.

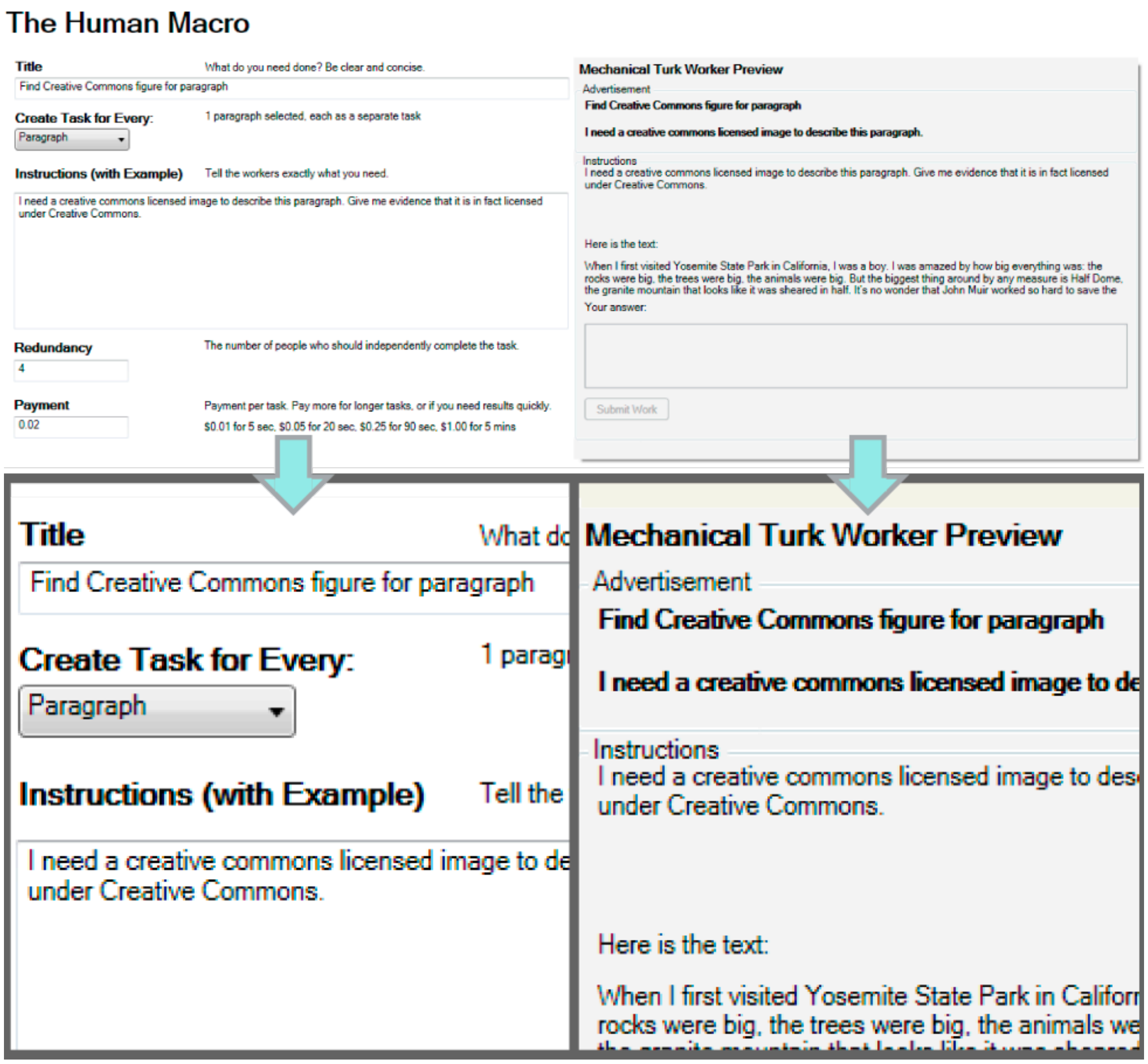
When the crowd is finished, Soylent calls out the erroneous sections with a purple dashed underline. If the user clicks on the error, a drop-down menu explains the problem and offers a list of alternatives. By clicking on the desired alternative, the user replaces the incorrect text with an option of his or her choice. If the user hovers over the Error Descriptions menu item, the popout menu suggests additional second-opinions of why the error was called out.

3.1.3 The Human Macro: Natural Language Crowd Scripting

Embedding crowd workers in an interface allows us to reconsider designs for short end-user programming tasks. Typically, users need to translate their intentions into algorithmic thinking explicitly via a scripting language or implicitly through learned activity [41]. But tasks conveyed to humans can be written in a much more natural way. While natural language command interfaces continue to struggle with unconstrained input over a large search space, humans are good at understanding written instructions.

The Human Macro is Soylent's natural language command interface. Soylent users can use it to request arbitrary work quickly in human language. Launching the Human Macro opens a request form (Figure 3-4). The design challenge here is to ensure that the user creates tasks that are scoped correctly for a Mechanical Turk worker. We wish to prevent the user from spending money on a buggy command.

The form dialog is split in two mirrored pieces: a task entry form on the left, and a preview of what the worker will see on the right. The preview contextualizes the user's request, reminding the user that they are writing something akin to a Help Wanted or Craigslist advertisement. The form suggests that the user provide an example input



The Human Macro

Title — What do you need done? Be clear and concise.

Find Creative Commons figure for paragraph

Create Task for Every: — 1 paragraph selected, each as a separate task

Paragraph

Instructions (with Example) — Tell the workers exactly what you need.

I need a creative commons licensed image to describe this paragraph. Give me evidence that it is in fact licensed under Creative Commons.

Redundancy — The number of people who should independently complete the task.

4

Payment — Payment per task. Pay more for longer tasks, or if you need results quickly.

0.02 — $0.01 for 5 sec, $0.05 for 20 sec, $0.25 for 90 sec, $1.00 for 5 mins

Mechanical Turk Worker Preview

Advertisement

Find Creative Commons figure for paragraph

I need a creative commons licensed image to describe this paragraph.

Instructions

I need a creative commons licensed image to describe this paragraph. Give me evidence that it is in fact licensed under Creative Commons.

Here is the text:

When I first visited Yosemite State Park in California, I was a boy. I was amazed by how big everything was: the rocks were big, the trees were big, the animals were big. But the biggest thing around by any measure is Half Dome, the granite mountain that looks like it was sheared in half. It's no wonder that John Muir worked so hard to save the

Your answer:

Title What do
Find Creative Commons figure for paragraph
Create Task for Every: 1 parag
Paragraph
Instructions (with Example) Tell the
I need a creative commons licensed image to de
under Creative Commons.
Mechanical Turk Worker Preview
Advertisement
Find Creative Commons figure for paragraph
I need a creative commons licensed image to de
Instructions
I need a creative commons licensed image to des
under Creative Commons.
Here is the text:
When I first visited Yosemite State Park in Califor
rocks were big, the trees were big, the animals we

Figure 3-4: The Human Macro allows users to request arbitrary tasks over their document. Left: user's request pane. Right: worker task preview, which updates as the user edits the request pane.

and output, which is an effective way to clarify the task requirements to workers. If the user selected text before opening the dialog, he has the option to split the task by each sentence or paragraph, so (for example) the task might be parallelized across all entries on a list. The user then chooses how many separate workers he would like to complete the task. The Human Macro helps debug the task by allowing a test run on one sentence or paragraph.

The user chooses whether the workers' suggestions should replace the existing text or just annotate it. If the user chooses to replace, the Human Macro underlines the text in purple and enables drop-down substitution like the Crowdproof interface. If the user chooses to annotate, the feedback populates comment bubbles anchored on the selected text by utilizing Word's reviewing comments interface.

3.2 Techniques for Programming Crowds

This section characterizes the challenges of leveraging crowd labor for open-ended document editing tasks. We introduce the Find-Fix-Verify pattern to improve output quality in the face of uncertain worker quality. As we prepared Soylent and explored the Mechanical Turk platform, we performed and documented dozens of experiments[2]. For this project alone, we have interacted with over 10,000 workers across over 2,500 different tasks. We draw on this experience in the sections to follow.

3.2.1 Challenges in Programming with Crowd Workers

We are primarily concerned with tasks where workers directly edit a user's data in an open-ended manner. These tasks include shortening, proofreading, and user-requested changes such as address formatting. In our experiments, it is evident that many of the raw results that workers produce on such tasks are unsatisfactory. As a rule-of-thumb, roughly 30% of the results from open-ended tasks are poor. This "30% rule" is supported by the experimental section of this paper as well. Clearly, a 30%

[2]`http://groups.csail.mit.edu/uid/deneme/`

error rate is unacceptable to the end user. To address the problem, it is important to understand the nature of unsatisfactory responses.

High Variance of Effort

Workers exhibit high variance in the amount of effort they invest in a task. We might characterize two useful personas at the ends of the effort spectrum, the *Lazy Worker* and the *Eager Beaver*. The Lazy Worker does as little work as necessary to get paid. For example, we asked workers to proofread the following error-filled paragraph from a high school essay site[3]. Ground-truth errors are colored below, highlighting some of the low quality elements of the writing:

> `The theme of loneliness features throughout many scenes in Of Mice and Men and is often the dominant theme of sections during this story. This theme occurs during many circumstances but is not present from start to finish. In my mind for a theme to be pervasive is must be present during every element of the story. There are many themes that are present most of the way through such as sacrifice, friendship and comradeship. But in my opinion there is only one theme that is present from beginning to end, this theme is pursuit of dreams.`

However, a Lazy Worker inserted only a single character to correct a spelling mistake. The single change is highlighted below:

> `The theme of loneliness features throughout many scenes in Of Mice and Men and is often the dominant theme of sections during this story. This theme occurs during many circumstances but is not present from start to finish. In my mind for a theme to be pervasive is must be present during every element of the story. There are many themes that are present most of the way through`

[3] `http://www.essay.org/school/english/ofmiceandmen.txt`

> `such as sacrifice, friendship and comradeship. But in my opinion there is only one theme that is present from beginning to end, this theme is pursuit of dreams.`

This worker fixed the spelling of the word *comradeship*, leaving many obvious errors in the text. In fact, it is not surprising that the worker chose to make this edit, since it was the only word in the paragraph that would have been underlined in their browser because it was misspelled.

A first challenge is thus to discourage workers from exhibiting such behavior. One approach to stopping Lazy Workers is the addition of clearly verifiable, quantitative questions (e.g., "How many sections does the Wikipedia article have?") that will force the Lazy Worker to complete the requester's task as a side effect of answering the verifiable task [102].

Equally problematic as Lazy Workers are Eager Beavers. Eager Beavers go beyond the task requirements in order to be helpful, but create further work for the user in the process. For example, when asked to reword a phrase, one Eager Beaver provided a litany of options:

> `The theme of loneliness features throughout many scenes in Of Mice and Men and is often the principal, significant, primary, preeminent, prevailing, foremost, essential, crucial, vital, critical theme of sections during this story.`

In their zeal, this worker rendered the resulting sentence ungrammatical. Eager Beavers may also leave extra comments in the document or reformat paragraphs. It would be problematic to funnel such work back to the user.

Both the Lazy Worker and the Eager Beaver are looking for a way to clearly signal to the requester that they have completed the work. Without clear guidelines, the Lazy Worker will choose the path that produces any signal and the Eager Beaver will produce too many signals.

Workers Introduce Errors

Workers attempting complex tasks can accidentally introduce substantial new errors. For example, when proofreading paragraphs about the novel *Of Mice and Men*, workers variously changed the title to just *Of Mice*, replaced existing grammar errors with new errors of their own, and changed the text to state that *Of Mice and Men* is a movie rather than a novel. Such errors are compounded if the output of one worker is used as input for other workers.

The Result: Low-Quality Work

These issues compound into what we earlier termed the 30% rule: that roughly a third of the suggestions we get from workers are not high-enough quality to show an end user. We cannot simply ask workers to help shorten or proofread a paragraph: we need to guide and coordinate their activities.

These two personas are not particular to Mechanical Turk. Whether we are using intrinsic or extrinsic motivators — money, love, fame, or others — there is almost always an uneven distribution of participation. For example, in Wikipedia, there are many Eager Beaver editors who try hard to make edits, but they introduce errors along the way and often have their work reverted [110]. Likewise, many Lazy participants on social networks will respond to content just to look engaged rather than because of any deep interest in the material [144].

3.2.2 The Find-Fix-Verify Pattern

Crowd-powered systems must control the efforts of both the Eager Beaver and Lazy Worker and limit introduction of errors. Absent suitable control techniques for open-ended tasks, the rate of problematic edits is too high to be useful. We feel that the state of programming crowds is analogous to that of UI technology before the introduction of design patterns like Model-View-Controller, which codified best practices. In this section, we propose the Find-Fix-Verify pattern as one method of programming

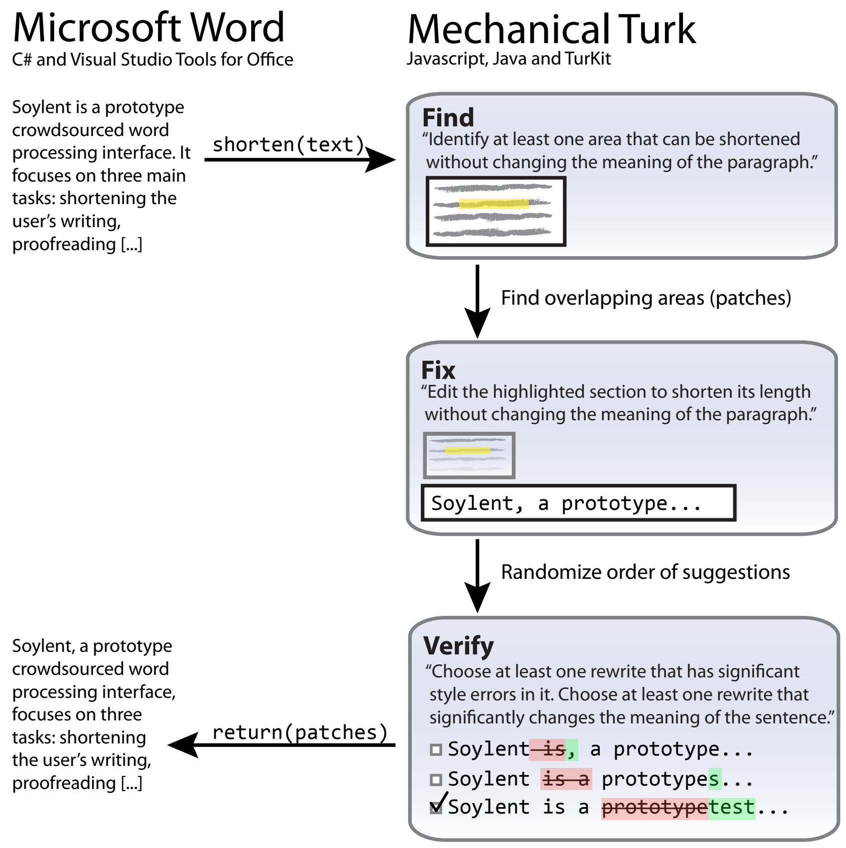

Microsoft Word
C# and Visual Studio Tools for Office
Soylent is a prototype crowdsourced word processing interface. It focuses on three main tasks: shortening the user's writing, proofreading [...]
shorten(text)
Mechanical Turk
Javascript, Java and TurKit
Find
"Identify at least one area that can be shortened without changing the meaning of the paragraph."
Find overlapping areas (patches)
Fix
"Edit the highlighted section to shorten its length without changing the meaning of the paragraph."
Soylent, a prototype...
Randomize order of suggestions
Verify
"Choose at least one rewrite that has significant style errors in it. Choose at least one rewrite that significantly changes the meaning of the sentence."
Soylent ~~is~~, a prototype...
Soylent ~~is a~~ prototypes...
Soylent is a ~~prototype~~test...
return(patches)
Soylent, a prototype crowdsourced word processing interface, focuses on three tasks: shortening the user's writing, proofreading [...]

Figure 3-5: Find-Fix-Verify identifies patches in need of editing, recruits workers to fix the patches, and votes to approve work.

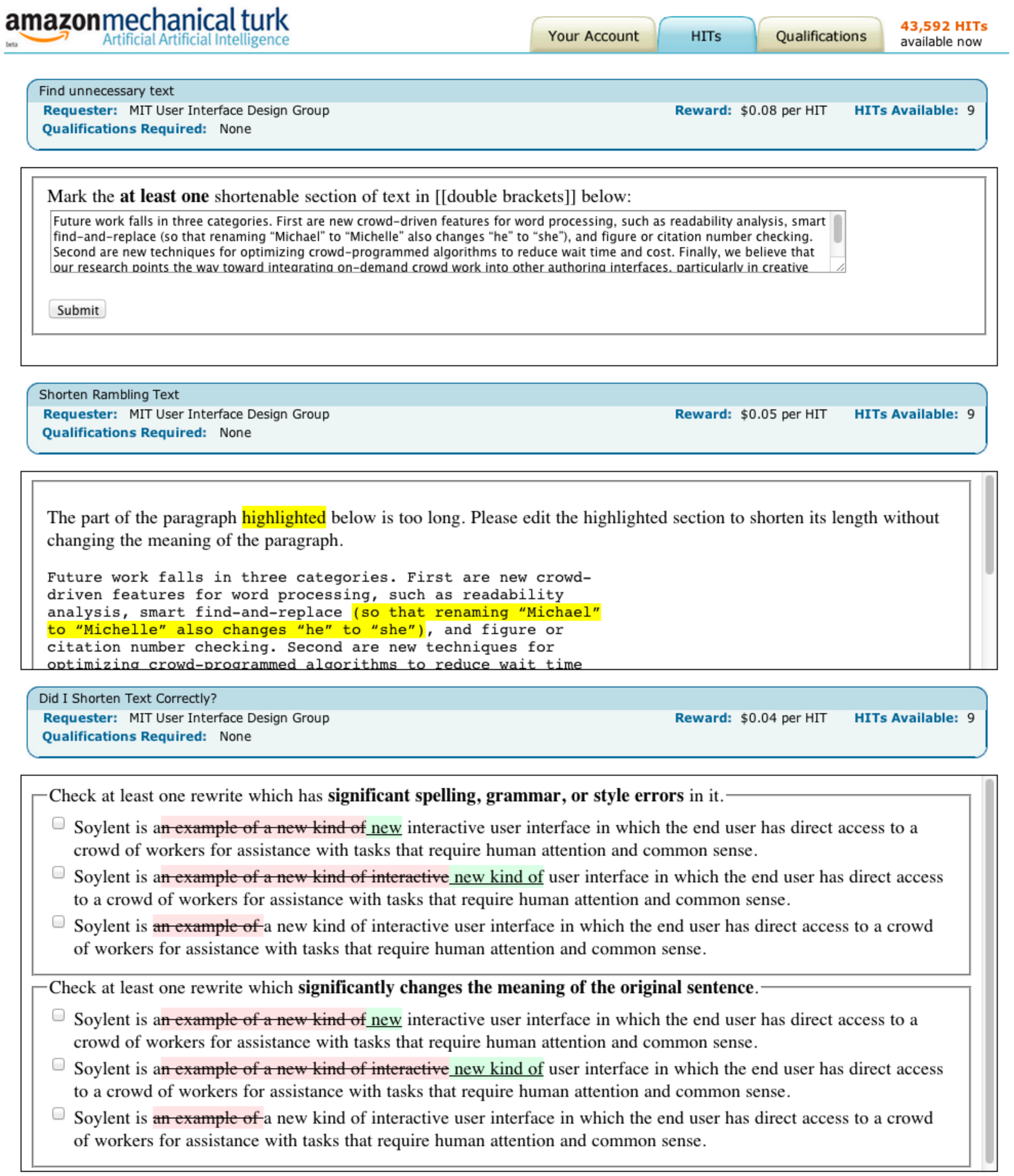
crowds to reliably complete open-ended tasks that directly edit the user's data[4]. We describe the pattern and then explain its use in Soylent across tasks like proofreading and text shortening.

Find-Fix-Verify Description

The Find-Fix-Verify pattern separates open-ended tasks into three stages where workers can make clear contributions. The workflow is visualized in Figure 3-5, and Figure 3-6 shows the Mechanical Turk tasks.

Both Shortn and Crowdproof use the Find-Fix-Verify pattern. We will use Shortn as an illustrative example in this section. To provide the user with near-continuous

[4]Closed-ended tasks like voting can test against labeled examples for quality control [124]. Open-ended tasks have many possible correct answers, so gold standard voting is less useful.



Find unnecessary text
Requester: MIT User Interface Design Group **Reward:** $0.08 per HIT **HITs Available:** 9
Qualifications Required: None

Mark the **at least one** shortenable section of text in [[double brackets]] below:

Future work falls in three categories. First are new crowd-driven features for word processing, such as readability analysis, smart find-and-replace (so that renaming "Michael" to "Michelle" also changes "he" to "she"), and figure or citation number checking. Second are new techniques for optimizing crowd-programmed algorithms to reduce wait time and cost. Finally, we believe that our research points the way toward integrating on-demand crowd work into other authoring interfaces, particularly in creative

Shorten Rambling Text
Requester: MIT User Interface Design Group **Reward:** $0.05 per HIT **HITs Available:** 9
Qualifications Required: None

The part of the paragraph highlighted below is too long. Please edit the highlighted section to shorten its length without changing the meaning of the paragraph.

```
Future work falls in three categories. First are new crowd-
driven features for word processing, such as readability
analysis, smart find-and-replace (so that renaming "Michael"
to "Michelle" also changes "he" to "she"), and figure or
citation number checking. Second are new techniques for
optimizing crowd-programmed algorithms to reduce wait time
```

Did I Shorten Text Correctly?
Requester: MIT User Interface Design Group **Reward:** $0.04 per HIT **HITs Available:** 9
Qualifications Required: None

Check at least one rewrite which has **significant spelling, grammar, or style errors** in it.

- ☐ Soylent is a~~n example of a new kind of~~ new interactive user interface in which the end user has direct access to a crowd of workers for assistance with tasks that require human attention and common sense.
- ☐ Soylent is a~~n example of a new kind of interactive~~ new kind of user interface in which the end user has direct access to a crowd of workers for assistance with tasks that require human attention and common sense.
- ☐ Soylent is ~~an example of~~ a new kind of interactive user interface in which the end user has direct access to a crowd of workers for assistance with tasks that require human attention and common sense.

Check at least one rewrite which **significantly changes the meaning of the original sentence**.

- ☐ Soylent is a~~n example of a new kind of~~ new interactive user interface in which the end user has direct access to a crowd of workers for assistance with tasks that require human attention and common sense.
- ☐ Soylent is a~~n example of a new kind of interactive~~ new kind of user interface in which the end user has direct access to a crowd of workers for assistance with tasks that require human attention and common sense.
- ☐ Soylent is ~~an example of~~ a new kind of interactive user interface in which the end user has direct access to a crowd of workers for assistance with tasks that require human attention and common sense.

Figure 3-6: Find-Fix-Verify proceeds in three stages. Tasks for each stage are compressed into one page here. The top task is Find, the middle one is Fix, and the bottom one is Verify.

control of paragraph length, Shortn should produce many alternative rewrites without changing the meaning of the original text or introduce[5] grammatical errors. We begin by splitting the input region into paragraphs.

The first stage, Find, asks several workers to identify patches of the user's work that need more attention. For example, when shortening, the Find stage asks ten workers for at least one phrase or sentence that needs to be shortened. Any single worker may produce a noisy result (e.g. Lazy workers might prefer errors near the beginning of a paragraph). The Find stage aggregates independent opinions to find the most consistently cited problems: multiple independent agreement is typically a strong signal that a crowd is correct. Soylent keeps patches where at least 20% of the workers agree. These are then fed in parallel into the Fix stage.

The Fix stage recruits workers to revise each agreed-upon patch. Each task now consists of a constrained edit to an area of interest. Workers see the patch highlighted in the paragraph and are asked to fix the problem (e.g., shorten the text). The worker can see the entire paragraph but only edit the sentences containing the patch. A small number (3–5) of workers propose revisions. Even if 30% of work is bad, 3–5 submissions are sufficient to produce viable alternatives. In Shortn, workers also vote on whether the patch can be cut completely. If so, we introduce the empty string as a revision.

The Verify stage performs quality control on revisions. We randomize the order of the unique alternatives generated in the Fix stage and ask 3–5 new workers to vote on them (Figure 3-5). We either ask workers to vote on the best option (when the interface needs a default choice, like Crowdproof) or to flag poor suggestions (when the interface requires as many options as possible, like Shortn). To ensure that workers cannot vote for their own work, we ban all Fix workers from participating in the Verify stage for that paragraph. To aid comparison, the Mechanical Turk task annotates each rewrite using color and strikethroughs to highlight its differences from

[5]Word's grammar checker, eight authors and six reviewers on the original Soylent paper did not catch the error in this sentence. Crowdproof later did, and correctly suggested that "introduce" should be "introducing".

the original. We use majority voting to remove problematic rewrites and to decide if the patch can be removed. At the end of the Verify stage, we have a set of candidate patches and a list of verified rewrites for each patch.

To keep the algorithm responsive, we use a 15-minute timeout at each stage. If a stage times out, we still wait for at least six workers in Find, three workers in Fix, and three workers in Verify.

Pattern Discussion

Why should tasks be split into independent Find-Fix-Verify stages? Why not let workers find an error and fix it, for increased efficiency and economy? Lazy Workers will always choose the easiest error to fix, so combining Find and Fix will result in poor coverage. By splitting Find from Fix, we can direct Lazy Workers to propose a fix to patches that they might otherwise ignore. Additionally, splitting Find and Fix enables us to merge work completed in parallel. Had each worker edited the entire paragraph, we would not know which edits were trying to fix the same problem. By splitting Find and Fix, we can map edits to patches and produce a much richer user interface—for example, the multiple options in Crowdproof's replacement dropdown.

The Verify stage reduces noise in the returned result. The high-level idea here is that we are placing the workers in productive tension with one another: one set of workers is proposing solutions, and another set is tasked with looking critically at those suggestions. Anecdotally, workers are better at vetting suggestions than they are at producing original work. Independent agreement among Verify workers can help certify an edit as good or bad. Verification trades off time lag with quality: a user who can tolerate more error but needs less time lag might opt not to verify work or use fewer verification workers.

Find-Fix-Verify has downsides. One challenge that the Find-Fix-Verify pattern shares with other Mechanical Turk algorithms is that it can stall when workers are slow to accept the task. Rather than wait for ten workers to complete the Find task before moving on to Fix, a timeout parameter can force our algorithm to advance if a minimum threshold of workers have completed the work. Find-Fix-Verify also makes it difficult for a particularly skilled worker to make large changes: decomposing the task makes it easier to complete for the average worker, but may be more frustrating for experts in the crowd.

3.3 Implementation

Soylent consists of a front-end application-level add-in to Microsoft Word and a back-end service to run Mechanical Turk tasks (Figure 3-5). The Microsoft Word plug-in is written using Microsoft Visual Studio Tools for Office (VSTO) and the Windows Presentation Foundation (WPF). Back-end scripts use the TurKit Mechanical Turk toolkit [129].

Shortn in particular must choose a set of rewrites when given a candidate slider length. When the user specifies a desired maximum length, Shortn searches for the longest combination of rewrites subject to the length constraint. A simple implementation would exhaustively list all combinations and then cache them, but this approach scales poorly with many patches. If runtime becomes an issue, we can view the search as a multiple-choice knapsack problem. In a multiple-choice knapsack problem, the items to be placed into the knapsack come from multiple classes, and only one item from each class may be chosen. So, for Shortn, each item class is an area of text with one or more options: each class has one option if it was not selected as a patch, and more options if the crowd called out the text as a patch and wrote alternatives. The multiple-choice knapsack problem can be solved with a polynomial time dynamic programming algorithm.

3.4 Evaluation

Our initial evaluation sought to establish evidence for Soylent's end-to-end feasibility, as well as to understand the properties of the Find-Fix-Verify design pattern. Full input texts for these evaluations are available in the appendix.

3.4.1 Shortn Evaluation

We evaluated Shortn quantitatively by running it on example texts. Our goal was to see how much Shortn could shorten text, as well as its associated cost and time characteristics. We collected five examples of texts that might be sent to Shortn, each between one and seven paragraphs long. We chose these inputs to span from preliminary drafts to finished essays and from easily understood to dense technical material (Table 3.1).

To simulate a real-world deployment, we ran the algorithms with a timeout enabled and set to twenty minutes for each stage. We required 6–10 workers to complete the Find tasks and 3–5 workers to complete the Fix and Verify tasks: if a Find task failed to recruit even six workers, it might wait indefinitely. To be slightly generous while matching going rates on Mechanical Turk, we paid \$0.08 per Find, \$0.05 per Fix, and \$0.04 per Verify.

Each resulting paragraph had many possible variations depending on the number of shortened alternatives that passed the Verify stage — we chose the shortest possible version for analysis and compared its length to the original paragraph. We also measured *wait time*, the time between posting the task and the worker accepting the task, and *work time*, the time between acceptance and submission. In all tasks, it was possible for the algorithm to stall while waiting for workers, having a large effect on averages. Therefore, we report medians, which are more robust to outliers.

Results

Shortn produced revisions that were 78%–90% of the original document length. For reference, a reduction to 85% could slim an $11\frac{3}{4}$ page ACM paper draft down to 10 pages with no substantial cuts in the content. Table 3.1 summarizes and gives examples of Shortn's behavior. Typically, Shortn focused on unnecessarily wordy phrases like "are going to have to" (Table 3.1, Blog). Workers merged sentences when patches spanned sentence boundaries (Table 3.1, Classic UIST Paper), and occasionally cut whole phrases or sentences.

Input Text	**Original Length**	**Final Length**	**Work Stats**	**Time per Paragraph**	**Example Output**
Blog	3 paragraphs, 12 sentences, 272 words	83% character length	$4.57, 158 workers	46–57 min	Print publishers are in a tizzy over Apple's new iPad because they hope to ~~finally~~ be able to charge for their digital editions. But in order to get people to pay for their magazine and newspaper apps, they ~~are going to~~ have to offer something different that readers cannot get at the newsstand or on the open Web.
Classic UIST Paper [92]	7 paragraphs, 22 sentences, 478 words	87%	$7.45, 264 workers	49–84 min	The metaDESK effort is part of the larger Tangible Bits project~~. The Tangible Bits vision paper~~, which introduced the metaDESK ~~along with~~and two companion platforms, the transBOARD and ambientROOM.
Draft UIST Paper [194]	5 paragraphs, 23 sentences, 652 words	90%	$7.47, 284 workers	52–72 min	~~In this paper we argue that~~ it is possible and desirable to combine the easy input affordances of text with the powerful retrieval and visualization capabilities of graphical applications. We present WenSo, ~~a tool that~~which uses lightweight text input to capture richly structured information for later retrieval and navigation in a graphical environment.
Rambling Enron E-mail	6 paragraphs, 24 sentences, 406 words	78%	$9.72, 362 workers	44–52 min	~~A previous board member,~~ Steve Burleigh~~,~~ created our web site last year and gave me alot of ideas. ~~For this year,~~ I found a web site called eTeamZ that hosts web sites for sports groups. Check out our new page: [. . .]
Technical Writing [5]	3 paragraphs, 13 sentences, 291 words	82%	$4.84, 188 workers	132–489 min	Figure 3 shows the pseudocode that implements this design for Lookup. FAWN-DS extracts two fields from the 160-bit key: ~~the i low order bits of the key~~ (the index bits) and the next 15 low order bits ~~(the key fragment)~~.

Table 3.1: Our evaluation run of Shortn produced revisions between 78%–90% of the original paragraph length on a single run. The Example Output column contains example edits from each input.

To investigate time characteristics, we separate the notion of wait time from work time. The vast majority of Shortn's running time is currently spent waiting, because it can take minutes or hours for workers to find and accept the task. Here, our current wait time — summing the median Find, median Fix, and median Verify — was 18.5 minutes (1st Quartile $Q_1 = 8.3$ minutes, 3rd Quartile $Q_3 = 41.6$ minutes). This wait time can be much longer because tasks can stall waiting for workers, as Table 3.1 shows. However, the next chapter will demonstrate techniques to reduce this wait time by several orders of magnitude.

Considering only work time and assuming negligible wait time, Shortn produced cuts within minutes. We again estimate overall work time by examining the median amount of time a worker spent in each stage of the Find-Fix-Verify process. This process reveals that the median shortening took 118 seconds of work time, or just under two minutes, when summed across all three stages ($Q_1 = 60$ seconds, $Q_3 = 3.6$ minutes). Using the recruitment techniques to come in Chapter 4, users may see shortening tasks approaching a limit of two minutes.

The average paragraph cost $1.41 to shorten under our pay model. This cost split into $0.55 to identify an average of two patches, then $0.48 to generate alternatives and $0.38 to filter results for each of those patches. Were we instead to use a $0.01 pay rate for these tasks, the process would cost $0.30 per paragraph. Our experience is that paying less slows down the later parts of the process, but it does not impact quality [141] — it would be viable for shortening paragraphs under a loose deadline.

Qualitatively, Shortn was most successful when the input had unnecessary text. For example, with the Blog input, Shortn was able to remove several words and phrases without changing the meaning of the sentence. Workers were able to blend these cuts into the sentence easily. Even the most technical input texts had extraneous phrases, so Shortn was usually able to make at least one small edit of this nature in each paragraph. As Soylent runs, it can collect a large database of these straightforward rewrites, then use them to train a machine learning algorithm to suggest some shortenings automatically.

Shortn occasionally introduced errors into the paragraph. While workers tended to stay away from cutting material they did not understand, they still occasionally flagged such patches. As a result, workers sometimes made edits that were grammatically appropriate but stylistically incorrect. For example, it may be inappropriate to remove the academic signaling phrase "In this paper we argue that..." from an introduction. Cuts were a second source of error: workers in the Fix stage would vote that a patch could be removed entirely from the sentence, but were not given the chance to massage the effect of the cut into the sentence. So, cuts often led to capitalization and punctuation problems at sentence boundaries. Modern auto-correction techniques could catch many of these errors. Parallelism was another source of error: for example, in Technical Writing (Table 3.1), the two cuts were from two different patches, and thus handled by separate workers. These workers could not predict that their cuts would not match, one cutting the parenthetical and the other cutting the main phrase.

To investigate the extent of these issues, we coded all 126 shortening suggestions as to whether they led to a grammatical error. Of these suggestions, 37 suggestions were ungrammatical, again supporting our rule of thumb that 30% of raw worker edits will be noisy. The Verify step caught 19 of the errors (50% of 37) while also removing 15 grammatical sentences. Its error rate was thus (18 false negatives + 15 false positives) / 137 = 26.1%, again near 30%. Microsoft Word's grammar checker caught 13 of the errors. Combining Word and Shortn caught 24 of the 37 errors.

We experimented with feeding the shortest output from the Blog text back into the algorithm to see if it could continue shortening. It continued to produce cuts between 70–80% with each iteration. We ceased after 3 iterations, having shortened the text to less than 50% length without sacrificing much by way of readability or major content:

> Print publishers are in a tizzy over Apple's iPad. We've seen interactive graphics, photo slide shows, and embedded videos.
>
> What should a magazine cover look like on the iPad? One way these covers could change is by using a video loop for the background image. Jesse

> Rosten, a photographer in California, created the video mockup below of what a cover of Sunset Magazine might look like on the iPad.
>
> The video shows ocean waves lapping a beach as typographical elements appear on the page almost like movie credits. This is just a mockup Rosten came up with on his own and The only way people are going to pay for these apps is if they create new experiences for readers.

The full-length, original text is in Appendix A.

3.4.2 Crowdproof Evaluation

To evaluate Crowdproof, we obtained a set of five input texts in need of proofreading (Table 3.2). We manually labeled all spelling, grammatical and style errors in each of the five inputs, identifying a total of 49 errors. We then ran Crowdproof on the inputs using a 20-minute stage timeout, with prices \$0.06 for Find, \$0.08 for Fix, and \$0.04 for Verify. We measured the errors that Crowdproof caught, that Crowdproof fixed, and that Word caught. We ruled that Crowdproof had caught an error if one of the identified patches contained the error.

Results

Soylent's proofreading algorithm caught 33 of the 49 errors (67%). For comparison, Microsoft Word's grammar checker found 15 errors (30%). Combined, Word and Soylent flagged 40 errors (82%). Word and Soylent tended to identify different errors, rather than both focusing on the easy and obvious mistakes. This result lends more support to Crowdproof's approach: it can focus on errors that automatic proofreaders have not already identified.

Crowdproof was effective at fixing errors that it found. Using the Verify stage to choose the best textual replacement, Soylent fixed 29 of the 33 errors it flagged (88%). To investigate the impact of the Verify stage, we labeled each unique correction that workers suggested as grammatical or not. Fully 28 of 62 suggestions, or 45%, were

Input Text	**Content**	**Errors** all/caught/fixed	**Workers**	**Time**	**Example Output**
Passes Word's Checker	1 paragraph, 4 sentences, 49 words	9 / 9 / 8	$4.76 77 workers	48 min	Marketing ~~are~~is bad for brands big and small. You ~~K~~know ~~W~~what I am ~~S~~saying. It is no wonder~~ing~~ that advertising~~s are~~ is bad for compan~~y~~ies in America, Chicago and Germany. Updating of brand image ~~are~~is bad for processes in one company and many companies.
English as a Second Language	1 paragraph, 8 sentences, 166 words	12 / 5 / 4	$4.72 79 workers	42–53 min	However, while GUI made using computers ~~be~~ more intuitive and easier to learn, it didn't ~~let people be able to~~ control computers efficiently. The masses only can use the software developed by software companies, unless they know how to write programs.
Notes from a Talk	2 paragraphs, 8 sentences, 107 words	14 / 8 / 8	$2.26 38 workers	47 min	~~Blah blah blah —~~This is an argument about whether there should be a standard "~~nosql~~NoSQL storage" API to protect developers storing their stuff in proprietary services in the cloud. ~~Probably unrealistic.~~ To protect yourself, use an open software offering, ~~and~~ self-host or go with hosting solution that uses open offering.
Wikipedia	1 paragraph, 5 sentences, 63 words	8 / 7 / 6	$2.18 36 workers	54 min	Dandu Monara (Flying Peacock, Wooden Peacock), The Flying ~~m~~Machine ~~able to fly~~. The King Ravana (Sri Lanka) built it. Accor~~in~~ding to ~~h~~Hindu belie~~ves~~fs in Ramayanaya King Ravana used "Dandu Monara" for abduct queen Seetha from Rama. According to believers, "Dandu Monara" landed at Werangatota.
UIST Draft [194]	1 paragraph, 6 sentences, 135 words	6 / 4 / 3	$3.30 53 workers	96 min	Many of these problems vanish if we turn to a much older recording technology - - ~~-~~text. When we enter text, each (pen or key) stroke is being used to record the actual information we care about~~- - -~~; none is wasted on application navigation or configuration.

Table 3.2: A report on Crowdproof's runtime characteristics and example output.

ungrammatical. The fact that such noisy suggestions produced correct replacements again suggests that workers are much better at verification than they are at authoring.

Crowdproof's most common problem was missing a minor error that was in the same patch as a more egregious error. The four errors that Crowdproof failed to fix were all contained in patches with at least one other error; Lazy Workers fixed only the most noticeable problem. A second problem was a lack of domain knowledge: in the ESL example in Table II, workers did not know what a GUI was, so they could not know that the author intended "GUIs" instead of "GUI". There were also stylistic opinions that the original author might not have agreed with: in the Draft UIST example in Table II, the author clearly had a penchant for triple dashes that the workers did not appreciate.

Crowdproof shared many running time characteristics with Shortn. Its median work time was 2.8 minutes (Q_1 = 1.7 minutes, Q_3 = 4.7 minutes), so it completes in very little work time. Similarly to Shortn, its wait time was 18 minutes (Median = 17.6, Q_1 = 9.8, Q_3 = 30.8). It cost more money to run per paragraph (μ = \$3.40, σ = \$2.13) because it identified far more patches per paragraph: we chose paragraphs in dire need of proofreading.

3.4.3 Human Macro Evaluation

We were interested in understanding whether end users could instruct Mechanical Turk workers to perform open-ended tasks. Can users communicate their intention clearly? Can workers execute the amateur-authored tasks correctly?

Method

We generated five feasible Human Macro scenarios (Table 3.3). We recruited two sets of users: five undergraduate and graduate students in our computer science department (4 male) and five administrative associates in our department (all female). We showed each user one of the five prompts, consisting of an example input and output pair. We purposefully did not describe the task to the participants so that

Task	**Quality**	**Example Request**	**Example Input**	**Example Output**
Tense $0.10 1 paragraph	CS: 100% intention (20% accuracy), Admin: 100% (40%), Author: 100% (60%)	Admin: "Please change text in document from past tense to present tense."	I gave one final glance around before descending from the barrow. As I did so, my eye caught something [. . .]	I give one final glance around before descending from the barrow. As I do so, my eye catches something [. . .]
Figure $0.20 1 paragraph	CS: 75% (75%), Admin: 75% (75%), Author: 60% (60%)	CS: "Pick out keywords from the paragrah like Yosemite, rock, half dome, park. Go to a site which hsa CC licensed images [. . .]"	When I first visited Yosemite State Park in California, I was a boy. I was amazed by how big everything was [. . .]	`http://commons.wikimedia.org/wiki/File:03_yosemite_half_dome.jpg`
Opinions $0.15 1 paragraph	CS: 100% (100%), Admin: 100% (100%), Author: 100% (100%)	CS: "Please tell me how to make this paragraph communicate better. Say what's wrong, and what I can improve. Thanks!"	Take a look at your computer. Think about how you launch programs, edit documents, and browse the web. Don't you feel a bit lonely? [. . .]	This paragraph needs an objective I feel like. [. . .] After reading I feel like there should be about five more sentences [. . .]
Citation Gathering $0.40 3 citations	CS: 75% (75%), Admin: 100% (100%), Author: 66% (40%)	Admin: "Hi, please find the bibtex references for the 3 papers in brackets. You can located these by Google Scholar searches and clicking on bibtex."	Duncan and Watts [Duncan and watts HCOMP 09 anchoring] found that Turkers will do more work when you pay more, but that the quality is no higher.	`@conference title=Financial incentives and [...], author=Mason, W. and Watts, D.J., booktitle=HCOMP 09`
List Processing $0.05 10 inputs	CS: 82% (82%), Admin: 98% (96%), Author: 91% (68%)	Admin: "Please complete the addresses below to include all informtion needed as in example below. [. . .]"	Max Marcus, 3416 colfax ave east, 80206	Max Marcus 3416 E Colfax Ave Denver, CO 80206

Table 3.3: The five tasks in the left column led to a variety of request strategies. Terse, typo-filled user requests still often led to success.

we would not influence how they wrote their task descriptions. We then introduced participants to The Human Macro and described what it would do. We asked them to write a task description for their prompt using The Human Macro. We then sent the description to Mechanical Turk and requested that five workers complete each request. In addition to the ten requests generated by our participants, one author generated five requests himself to simulate a user who is familiar with Mechanical Turk.

We coded results using two quality metrics: intention (did the worker understand the prompt and make a good faith effort?) and accuracy (was the result flawless?). If the worker completed the task but made a small error, the result was coded as good intention and poor accuracy.

Results

Users were generally successful at communicating their intention (Table 3.3). The average command saw an 88% intention success rate (max = 100%, min = 60%). Typical intention errors occurred when the prompt contained two requirements: for example, the Figure task asked both for an image and proof that the image is Creative Commons-licensed. Workers read far enough to understand that they needed to find a picture, found one, and left. Successful users clearly signaled Creative Commons status in the title field of their request.

With accuracy, we again see that roughly 30% of work contained an error. (The average accuracy was 70.8%.) Workers commonly got the task mostly correct, but failed on some detail. For example, in the Tense task, some workers changed all but one of the verbs to present tense, and in the List Processing task, sometimes a field would not be correctly capitalized or an Eager Beaver would add too much extra information. These kinds of errors would be dangerous to expose to the user, because the user might likewise not realize that there is a small error in the work.

3.4.4 Impact of Price on Wait Time

The most obvious lever that users have is the ability to name a price. For interactive purposes, it is important to understand how price impacts wait time and work time. Can the user trade off time for money, getting faster results by paying more? There is evidence that offering more money means that workers complete more tasks, resulting in faster completion times for large batch tasks [141, 202]. How does this play out in a system like Soylent, where recruitment wait time dominates?

Method

To investigate the impact of price on time, we ran Crowdproof many times at different price points, measuring wait time and work time. We used the same input each time: the first paragraph from the ESL input. We isolated the Find stage from the Fix and Verify stages so that the Fix/Verify combo would always begin with the same patches. So, we ran iterations of two separate algorithms: Find, and Fix+Verify.

We chose price points of \$0.01, \$0.05, \$0.10, and \$0.50 to cover a range of prices on Mechanical Turk. Each price dictated the reward for a Find or a Fix task; Verify votes were always \$0.03. We fixed the price of Verify so that we could isolate the impact of price on Fix, rather than a combination of Fix+Verify, and get some notion of how much Mechanical Turk fluctuated.

Results

Investigating wait time, our most striking finding was that a small number of workers streamed into the task as soon as it was posted, no matter the price. On the Find task, all price points attracted at least one worker in the first 100 seconds, while the task was highly visible on the Mechanical Turk task listing when sorted by recency. All price points likewise attracted three workers in the first ten minutes. After about 15 minutes, the flow of new workers becomes a crawl, looking linear on an exponential plot. At this point, price began to have an impact. \$0.01 and \$0.05 behaved similarly, while \$0.10 and \$0.50 attracted workers more quickly. We can see this process in

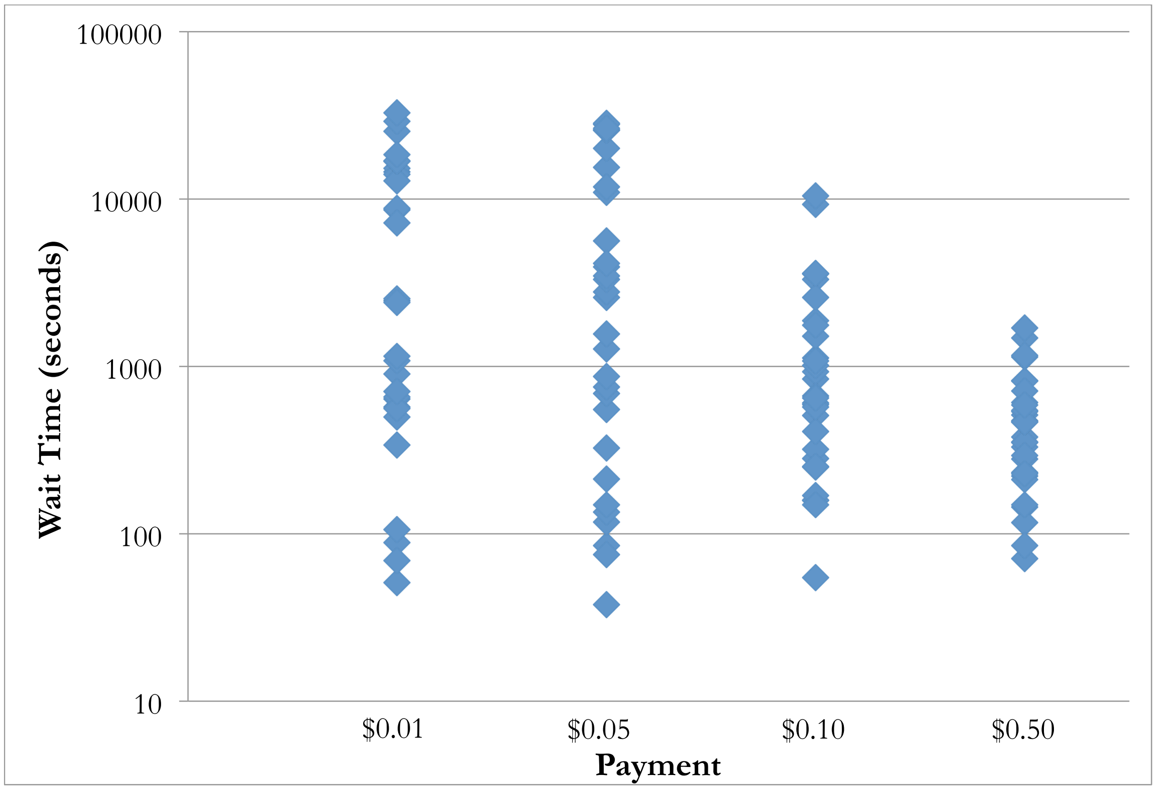


Figure 3-7: A group of workers always accepted the Find or Fix task while it was on the homepage, roughly within two–three minutes of posting. Raising the price sped up the arrival of the rest of the workforce, but did not impact these early workers.

Figure 3-7: the bottom parts of the columns in the scatterplot look similar, but \$0.10 and \$0.50 pull in the tail of workers that extends beyond 10,000 seconds for \$0.01 and \$0.05. This result suggests that for interactive applications that require low lag and a small number of workers, a low price will work. However, to attract a larger group of workers, we either need to be willing to wait or pay more.

The Fix task showed the same trends, though slightly accelerated because each worker could complete two Fix tasks at a time. Again, \$0.01 and \$0.05 performed similarly while \$0.10 and \$0.50 were slower.

These results suggest that the system may want to dynamically manage its wait time to get the most workers and exit before expected wait time begins growing exponentially. They also suggest that within the first few minutes of a task being posted to Mechanical Turk, the labor supply is relatively inelastic with respect to price. The market becomes more elastic at larger time scales.

In the next chapter, we will demonstrate ways to reduce this wait time by several orders of magnitude to create realtime crowds.

3.5 Discussion

While Soylent uses exclusively crowd contributions, it will be important to close the loop and integrate machine learning solutions. Soylent users are indirectly paying for a very large training corpus for sentence compression algorithms. This corpus contains patches that can be shortened, as well as multiple potential rewrites for each patch. As these algorithms improve, Shortn can propose some cuts automatically and go to the crowd only for the uncertain or harder patches. Ultimately, this hybrid crowd-AI system will scale much more successfully than just crowds.

Other issues with Soylent include ethics, privacy, and scale. We will return to these discussions in Chapter 7.

3.6 Conclusion

The following conclusion was Shortn'ed to 85% length:

This chapter presents Soylent, a word processing interface that uses crowd workers to help with proofreading, document shortening, editing and commenting tasks. Soylent is a~~n example of a~~ new~~kind of~~ interactive user interface in which the end user has direct access to a crowd of workers for assistance with tasks that require human attention and common sense. Implementing these ~~kinds of~~ interfaces requires new software programming patterns~~ for interface software~~, since crowds behave differently than computer systems. We have introduced one important pattern, Find-Fix-Verify, which splits complex editing tasks into a series of identification, generation, and verification stages ~~that use independent agreement and voting~~ to produce reliable results. We evaluated Soylent with a range of editing tasks, finding and correcting 82% of grammar errors~~ when combined with automatic checking~~, shortening text to approximately 85% of original length per iteration, and executing a variety of human macros successfully.

Future work falls in three categories. First are new crowd-driven features for word processing, such as readability analysis, smart find-and-replace ~~(so that renaming "Michael" to "Michelle" also changes "he" to "she")~~, and figure or citation number checking. Second are new techniques for optimizing crowd-programmed algorithms to reduce wait time and cost. Finally, we believe that our research points the way toward integrating on-demand crowd work into other authoring interfaces, particularly in creative domains ~~like image editing and programming~~.

Chapter 4

Realtime Crowdsourcing: Platform and Model

Crowd-powered systems such as Soylent demonstrate the potential to create new kinds of interactive applications, but these applications are limited by the problem of crowd latency. Crowdsourcing is only a reasonable choice if the user can wait significant lengths of time for a response. Existing "nearly realtime" techniques produce a single, unverified answer to a question in 56 seconds on average [21]. More complex workflows such as Find-Fix-Verify require roughly twenty minutes of wait and work time (Section 3.4.1).

Users are not used to waiting, and will abandon interfaces that are slow to react. Search engine usage decreases linearly as delays grow [173], and Jakob Nielsen argues that ten seconds is the maximum delay before a user loses focus on the interaction dialogue [150]. The much longer delays with crowdsourcing make it difficult for crowds to help with tasks in the moment-to-moment workflow. For example, using crowds to create a smarter copy-paste command would be difficult: a one minute wait between copying and pasting is unusable. We need new approaches if we want to realize the vision of the user pushing a button and seeing a crowd-powered result just seconds later.[1]

[1]This chapter has adapted, updated, and rewritten content from papers at UIST 2011 [16] and Collective Intelligence 2012 [17].

So, our goal is *realtime crowdsourcing*: completing non-trivial crowdsourced computation within seconds of the user's request — fast enough to feed back into an interface before the user loses focus. Realtime crowdsourcing can open up a broad new class of applications for crowd-powered interfaces.

The core contribution of this chapter is platform support for realtime crowds and a mathematical model of that platform.

To recruit realtime synchronous crowds, we present the *retainer model*. It hires crowd members in advance, then places them on hold for low cost and alerts them when a task is ready. Our most effective design results in 50% of workers returning within two seconds and 75% within three seconds. The retainer model's performance is striking in that it approaches human limits on the cognitive recognize-act cycle and motor reaction times [30]. It nearly zeroes out wait times, which in previous work ranged from twenty seconds [21] to twenty minutes (Chapter 3). Most importantly, however, it makes on-demand synchronous crowds available as a new resource for crowdsourcing.

Having introduced the retainer model, I will then turn to a mathematical model of retainer recruitment. This model allows us, for the first time, to understand how these approaches would work at large scale and to optimize the tradeoffs between retainer pool size, cost, and response time.

The mathematical model analyzes retainer recruitment using queueing theory [69] to understand the retainer model's performance at scale, in particular the trade-off between expected wait time and cost. We introduce a simple algorithm for choosing the optimal size of the retainer pool to minimize total cost to the requester subject to the requester's performance requirements: bounded wait time or bounded probability of missing a request. We then propose several improvements to the retainer model that reduce expected wait time. First, *retainer subscriptions* allow workers to sign up for push notifications for recruitment, which reduces the length of time it takes to recruit new workers onto retainer. Second, *combining retainer pools* across requesters leads to both cost and wait time improvements. Large retainer pools can then be made more effective by using *task routing* to connect appropriate workers to the tasks that need them. Third, a *precruitment* strategy recalls workers from retainer a few moments before a task is expected to arrive, dramatically lowering response time. We perform an early empirical evaluation demonstrating that precruitment results in median response times of just 500 milliseconds.

Our analysis carries several benefits. First, realtime tasks can now directly minimize their cost for a given performance requirement. Second, the retainer subscriptions allows workers to register for the tasks they like best and have them delivered, rather than constantly seeking out new work. Third, we demonstrate empirically that these techniques can overcome the retainer model's original limits of "crowds in two seconds" to deliver the feedback to the user within 500 milliseconds—finally under the one-second cognitive threshold for an end-user to remain in flow [150].

4.1 The Retainer Model

To power realtime applications like Adrenaline, we need to gather not just one individual but a small crowd quickly: a synchronous, flash crowd. We would like the crowd to turn their attention to our task as soon as it is available, for the system to spend as little money as possible.

Previous work, called quikTurKit, returns a single response roughly sixty seconds after a request [21]. To do so, it repeatedly lists new tasks on Amazon Mechanical Turk, regardless of whether a task is ready. If there are no pending tasks, quikTurKit shows the workers old tasks to keep them busy.

While it was clear from our explorations that workers would respond more quickly when paid more money (Section 3.4.4), it was not certain that they could respond fast enough for an interactive system. How should a retainer system be designed to 1) guarantee a fast response time, 2) be cheap enough to scale, and 3) maintain that response time after a long wait?

In this section, we introduce the *retainer model* for synchronous crowdsourcing and empirically derived design guidelines for its use. This approach solves all three issues by placing workers on retainer — signed up to do work when it is available —

for a small fee, then allowing them to pursue other work while they wait. When the user makes a request, the retainer model alerts the workers. Our designs result in a majority of workers returning two seconds after request. These workers arrive at the same time, enabling synchronous crowds.

4.1.1 Retainer Design and Wait Time

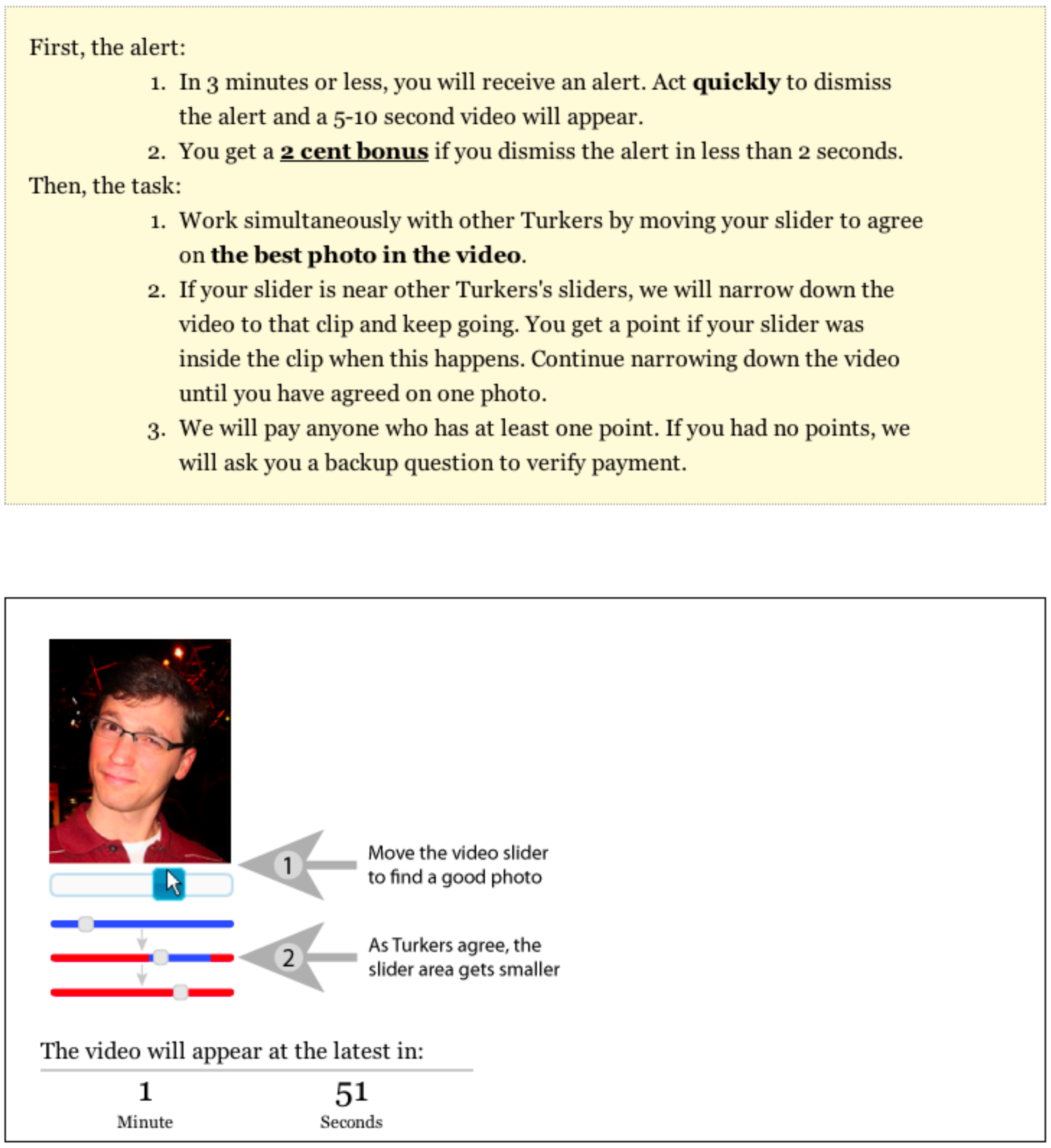
Workers agree to be put on retainer by accepting the task. They are given task instructions and an example, and told that they will be alerted when a task is ready (Figure 4-1). We scale the task price up by expected wait time, usually 0.5¢ per expected minute on retainer. Workers accept the price and maximum wait time up front. For example, with a 5¢ base task price, 0.5¢ per minute retainer, and maximum wait time of four minutes, the offered price is $5¢ + 4(0.5¢) = 7¢$. After workers agree, they are free to leave the browser tab open and pursue other work.

The worker's browser polls a work server to see if tasks are available. When a task is ready, the work server notifies the client, and the client's browser issues a Javascript `alert()` and an audio chime to signal the worker (Figure 4-2). Optionally, the work server may also offer a small bonus to reward quick responses, for example 3¢ to return within two seconds. Workers dismiss the alert when they arrive, then begin the task. If no new tasks are ready by the end of the retainer period, the retainer model gives workers an old task to perform, like quikTurKit [21]. As work arrives more consistently, however, the chance of wasting a task becomes lower.

4.1.2 Retainer Field Experiments

Can workers react quickly enough to support a realtime application, especially when they may be distracted with other tasks? This section describes field experiments of the retainer model that investigate its effectiveness.

Our high-level experimental approach was to vary the retainer time that workers would wait before seeing a task, and the design of the alert mechanism, then measure the latency between when the task was ready and when the worker dismissed the alert



First, the alert:

1. In 3 minutes or less, you will receive an alert. Act **quickly** to dismiss the alert and a 5-10 second video will appear.
2. You get a **2 cent bonus** if you dismiss the alert in less than 2 seconds.

Then, the task:

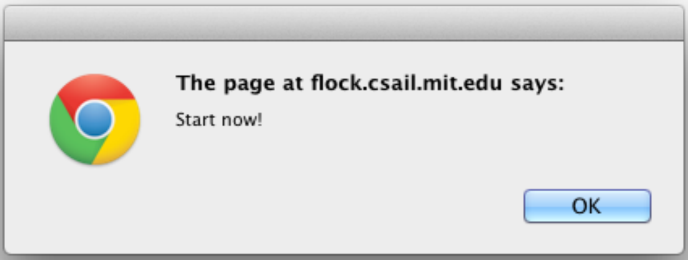
1. Work simultaneously with other Turkers by moving your slider to agree on **the best photo in the video**.
2. If your slider is near other Turkers's sliders, we will narrow down the video to that clip and keep going. You get a point if your slider was inside the clip when this happens. Continue narrowing down the video until you have agreed on one photo.
3. We will pay anyone who has at least one point. If you had no points, we will ask you a backup question to verify payment.

1
Move the video slider to find a good photo
2
As Turkers agree, the slider area gets smaller

The video will appear at the latest in:

1 Minute 51 Seconds

Figure 4-1: When a worker accepts a task with a retainer, the system displays an explanation of the maximum retainer time as well as a preview of the task that will eventually appear.


The page at flock.csail.mit.edu says:

Start now!

Figure 4-2: When a task arrives, a Javascript `alert()` draws the worker's attention to the application tab so they can begin.

to begin the task. We created a benchmark Mechanical Turk task that instructed workers to click on all the verbs in a random paragraph from a blog or a book. Workers were told that the task would be ready within a specific retainer time limit, then the web page began an invisible countdown that sampled uniformly between zero seconds and the maximum retainer time. So, for a five-minute retainer time, the average wait time was 2.5 minutes. When the countdown finished, the page alerted the worker and showed a task. We prevented workers from accepting more than one of our tasks at a time.

Study 1: Retainer Time

To test how long we could keep workers primed, we experimentally manipulated retainer time to vary between 0.5, 1, 2, 5, 10, and 30 minutes. We scaled payment linearly with retainer time, $\lfloor 2¢ + 1¢(\text{wait time}) \rfloor$: 2¢, 3¢, 4¢, 7¢, 12¢, and 32¢. We hypothesized that worker response time would increase after 1–2 minutes, as workers stopped monitoring the page.

To reduce the chance that workers would see multiple price points for the same task, we posted each set at different hours. We ran the experiment over a period of six days, in six separate one-hour periods each day, and randomized the order of conditions. A total of 280 workers completed 1545 tasks. We removed and rejected

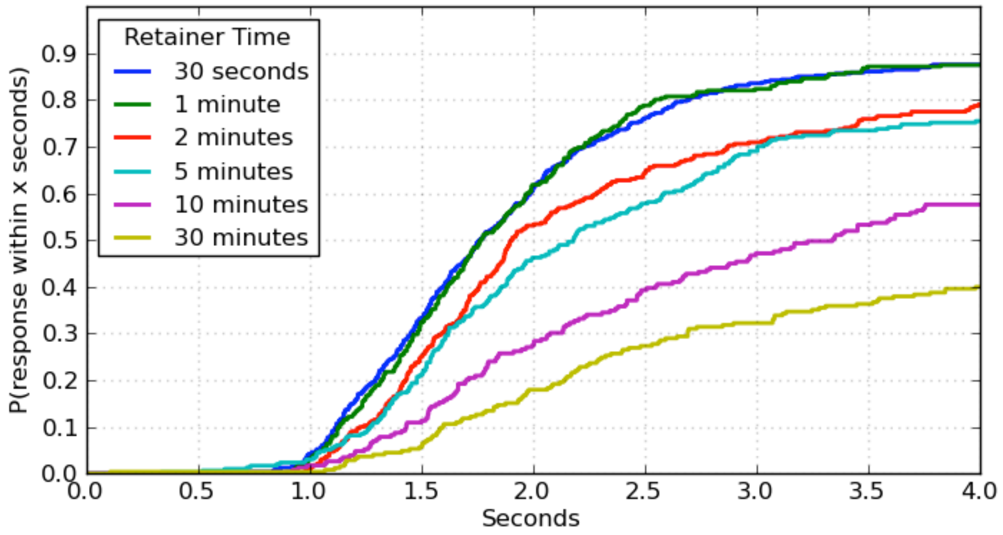


Figure 4-3: For retainer times under ten minutes, a majority of workers responded to the alert within two seconds and three-quarters responded within three seconds. N=1442.

	30 sec	**1 min**	**2 min**	**5 min**	**10 min**	**30 min**
Median	1.77 s	1.77 s	1.91 s	2.18 s	3.34 s	10.32 s
3rd quartile Q_3	2.44 s	2.39 s	3.46 s	3.75 s	—	—
Completion	86.6%	87.2%	82.9%	75.1%	66.4%	49.4%

Table 4.1: A tabular representation of Figure 4-3.

103 tasks because they disagreed significantly with our ground truth.

Results. For retainer times under ten minutes, 46–61% of workers dismissed the alert within two seconds and 69–84% of workers dismissed the alert within three seconds (Figure 4-3, Table 4.1). These curves in Figure 4-3 asymptote to a completion rate of 83–87%: the rest of the workers never returned to complete the task. Retainer times of ten minutes or more resulted in much lower completion rates, 49–66%. The median time between dismissing the alert and completing the first incremental piece of work (clicking on a verb) was 3.35 seconds across all conditions.

These results suggest that for wait times under ten minutes, we could expect to produce a crowd in two seconds, and a larger crowd in three seconds. In the next study, we investigate how to improve response time and completion rates further through retainer designs.

Study 2: Alert Design

While Study 1's results are already good enough to get a crowd quite quickly, can we improve on them by changing the reason that workers would pay attention? Can we incentivize the slow workers to move more quickly?

We investigated design and financial incentives to shift the curve so that more workers came within the first 2–3 seconds. We used the 12¢ 10-minute retainer condition from Study 1, which exhibited a low completion rate and a slower arrival rate. The *alert* condition functioned as in Study 1, with a Javascript alert and audio chime. Bonuses can be powerful incentives [141], so we designed a *reward* condition that paid workers a 3¢ bonus if they dismissed the alert within two seconds. Two seconds is short enough to be challenging, but not so short as to be out of reach. To keep workers' attention on the page, we created a *game* condition that let workers optionally play Tetris during the waiting period. Finally, to isolate the effectiveness of the Javascript alert, we created a *baseline* condition that displayed a large Go button on the page when the timer expired but did not use an audio or Javascript alert. We hypothesized that the bonus and game conditions might improve response time and completion rate.

For Study 2, we implemented a between-subjects design by randomly assigning each worker to a condition for the same verb-selection task. We posted tasks for four hours per day over four days. Workers completed 1913 tasks — we removed 90 for poor work quality.

Results. Paying a small reward for quick reactions had a strong positive impact on response time (Figure 4-4, Table 4.2). In the reward condition, 61% of workers responded within two seconds vs. 25% in the alert condition, and 74% responded within three seconds vs. 50% in the alert condition. Roughly speaking, the ten-minute retainer with reward had similar performance to the two-minute retainer without reward. In addition, workers in the reward condition completed 2.25 times as many tasks as those in the alert condition (734 vs. 325), suggesting that the small bonus has a disproportionately large impact on work volume. Predictably, the baseline condition

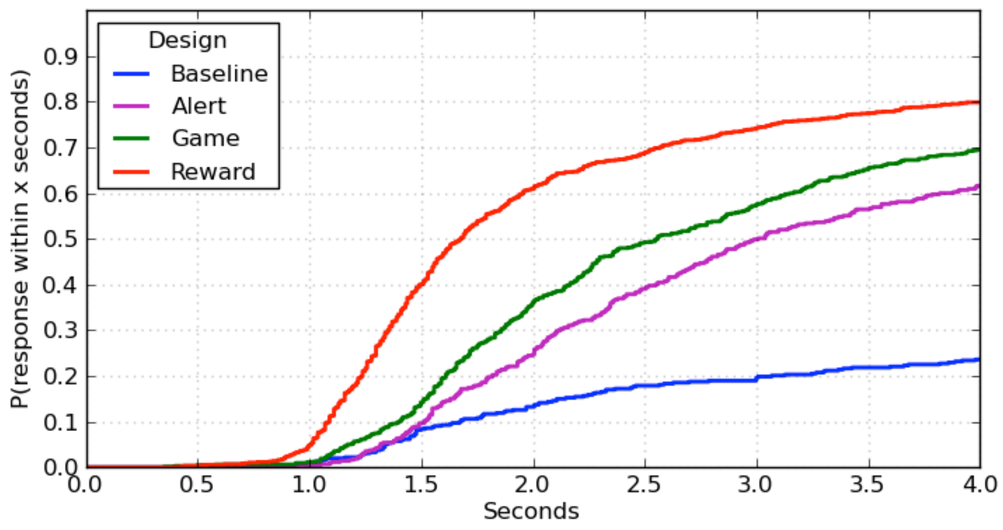


Figure 4-4: A small reward for fast response (red) led workers in a ten-minute retainer to respond as quickly as those on a two-minute retainer without reward (Figure 4-3, red). Other conditions included no alert (blue), an alert without bonus payment (purple), and a game to keep workers entertained (green). N=1913.

	Baseline	**Alert**	**Game**	**Reward**
Median	36.66 s	3.01 s	2.55 s	1.68 s
3rd quartile Q_3	—	6.92 s	5.01 s	3.07 s
Completion	64.2%	76.5%	76.7%	85.5%

Table 4.2: A tabular representation of Figure 4-4.

without the alert dialog performed poorly, with roughly 15% returning within two seconds. The game was not very popular (5.7% of completed tasks cleared a row in Tetris), but had a small positive impact on reaction times.

Retainer Model Discussion

Our data suggest that the retainer model can summon a crowd two seconds after the request is made. In exchange for a small fee, the retainer model recalls 50% of its workers in two seconds and 75% in three seconds. Though reaction times worsen as the retainer time increases, a small reward for quick response negates the problem. Our experiment commonly produced 10–15 workers on retainer at once, suggesting that users could fairly reliably summon a crowd of ten within three seconds. Applications with an early indication that the user will want help (for example, a mouseover on the feature's toolbar icon or an "Are You Sure?" dialog) can eliminate even this delay by alerting workers in advance. Section 4.4.3 will formalize this idea into a technique called *precruitment*.

The cost of the retainer model is attractive because it pays workers a small amount to wait, rather than spending money to repeat old tasks. The next section will quantify the cost of the retainer model more precisely and relate cost to performance guarantees.

4.2 Queueing Theory Model

Up to this point, the retainer model has not been optimized for cost or performance, nor do requesters have any analytic framework to understand the relationship between retainer pool size, cost, and response time. This section will develop an analytical, mathematical model of retainer recruitment. In particular, we can adapt queueing theory [69] to understand its performance at scale. This model leads to a simple algorithm for choosing the optimal size of the retainer pool to minimize total cost to the requester subject to the requester's performance requirements: maximum expected wait time or maximum probability of missing a request.

4.2.1 Model Formalization

In this section, we investigate a mathematical model of retainers. This model allows us to predict how long realtime tasks will need to wait. Suppose that the task maintains a set of retainer workers. When a task comes in, a worker leaves the retainer pool to work on the task and the retainer system recruits another worker to refill the pool. The goal is to maintain a large enough pool of retainer workers to handle incoming tasks. In other words, we want to minimize the probability that the retainer pool will be empty (no retainer workers left), subject to cost constraints. The risk is that a burst of task arrivals may exhaust the retainer pool before we can recruit replacement workers.

We will model this problem using queueing theory. In queueing theory, a set of *servers* are available to handle *jobs* as they arrive. If all servers are busy handling a job when a new job arrives, that job enters a queue of waiting tasks and is serviced as soon as it reaches the front of the queue. In our scenario, tasks are jobs, and retainer workers are servers.

Here, we will consider a class of algorithms that set an optimal retainer pool size. Suppose the retainer pool is c workers. As jobs come in and remove workers from the retainer pool, assume that the system always puts out enough requests for new workers to bring the pool back to c. That is, if there are c_0 workers in the pool, the system has issued $c - c_0$ outstanding requests. If, when a job arrives, the pool is empty, the system sets it aside for special processing: it directly recruits a worker, not for the pool, but for that job. In effect, a user with a diverted job is immediately alerted that the system is over capacity and the job will be handled out-of-band after a short delay. This final assumption may not accurately reflect how a running system would work, but it provides an upper bound on expected wait time and makes it easier to analyze the probability that a task will be serviced in realtime.

Suppose that tasks arrive as a Poisson process at rate λ, and retainer workers arrive after they are requested as a Poisson process at rate μ.[2] Then, the empty spots in the retainer pool, each of which will become filled when a worker arrives, can be thought of as busy machines occupied with a job whose completion time is a Poisson process with rate μ. In our setup, we also divert jobs that arrive when all machines are busy.

In other words, this is an M/M/c/c queue where jobs arrive at rate λ and have processing time μ. A basic M/M/1 queue assumes Poisson arrival and completion processes, a single server, and a potentially infinite queue. An M/M/c/c queue has c parallel machines instead of one, and rejects or redirects requests when there are no servers to immediately handle the incoming request [69]. Imagine a telephone system,

[2]These assumptions are perhaps overly ideal. Job arrivals on Mechanical Turk are heavy-tailed [89]. However, much of our analysis is independent of the arrival distribution, and systems can always substitute empirically observed distributions and solve numerically.

for example, that gives a busy signal if all c lines are busy. The meaning of μ has now changed slightly to indicate worker recruitment time instead of a job completion time.[3]

To optimize performance, we need to understand the probability that all workers are busy, since that is the case where a job has to wait (for expected time $1/\mu$). We also need to understand the cost of having a retainer pool of size c. Since the system pays workers proportional to how long they are on retainer without a job, the total cost is proportional to the average number of *idle* machines—these are the ones representing workers waiting on retainer. Finally, we will integrate worker abandonment into our model, since not all workers respond to the retainer alert.

Probability of an Empty Pool

The probability that a job must wait can be derived using Erlang's loss formula [69]. We set ρ, the *traffic intensity*, to be the ratio of the incoming task rate to the worker recruitment rate: $\rho = \lambda/\mu$. In M/M/c/c queueing systems, as we will demonstrate, $\rho < c$ is necessary for the system to keep up with incoming requests.

The probability of an empty retainer pool (all c "servers" busy) is Erlang's loss formula:

$$\pi(c) = \frac{\rho^c/c!}{\sum_{i=0}^{c} \rho^i/i!} \tag{4.1}$$

A remarkable property of Erlang's loss formula is that this relationship requires no assumptions about the distributions of job arrival time or worker recruitment time, in particular whether they are Poisson. It only depends on the means μ and λ.

Expected Waiting Time

For some applications, the probability of a task needing to wait is less important than the expected wait time for the task. The two quantities are directly related.

[3]This new μ is slightly counterintuitive: typical queueing systems have to wait for $1/\mu$ seconds on average for a server to complete a task. This queueing system does not wait at all, and instead reaches into the crowd to immediately recruit a new worker (server). So, the $1/\mu$ wait time refers to recruitment speed, not server work time. The mathematical analysis remains the same.

The expected wait time is the probability of an empty retainer pool multiplied by the expected wait time when the pool is empty. When the pool is empty, the requester recruits a new worker specifically for the task at rate μ, so the expected wait time in this case is $\frac{1}{\mu}$. So, the overall expected wait time is $\frac{1}{\mu}\pi(c)$:

$$\frac{1}{\mu}\pi(c) = \frac{1}{\mu}\frac{\rho^c/c!}{\sum_{i=0}^{c}\rho^i/i!} \tag{4.2}$$

This expression gives us a direct relationship between the size of the retainer pool, the arriving task and worker rates, and the expected wait time.

As a sanity check: when $\lambda \to 0$ (few arrivals) we have $\rho \to 0$ in which case $\pi(c) \to 0$.[4] In other words, we are very unlikely to have an empty pool so the expected wait time also goes to zero. When $\lambda \to \infty$ (many arrivals) the pool is almost certainly empty ($\pi(c) \to 1$), so all tasks must wait, and the expected wait time is $1/\mu$. This relationship is visualized in Figure 4-5(c).

Expected Cost

Once we understand expected waiting time, we can analyze the retainer model's cost characteristics. The earlier experiments suggested that workers could be maintained on retainer for \$0.30 per hour ($\frac{1}{2}$¢ per minute), but this analysis is fairly simplistic. To understand cost more completely, we need to know the expected number of workers on retainer.

The probability of having i busy servers in an M/M/c/c queue is a more general version of Erlang's loss formula:

$$\pi(i) = \frac{\rho^i/i!}{\sum_{i=0}^{c}\rho^i/i!} \tag{4.3}$$

[4]Actually, $\pi(c) \to \rho^c/c!$

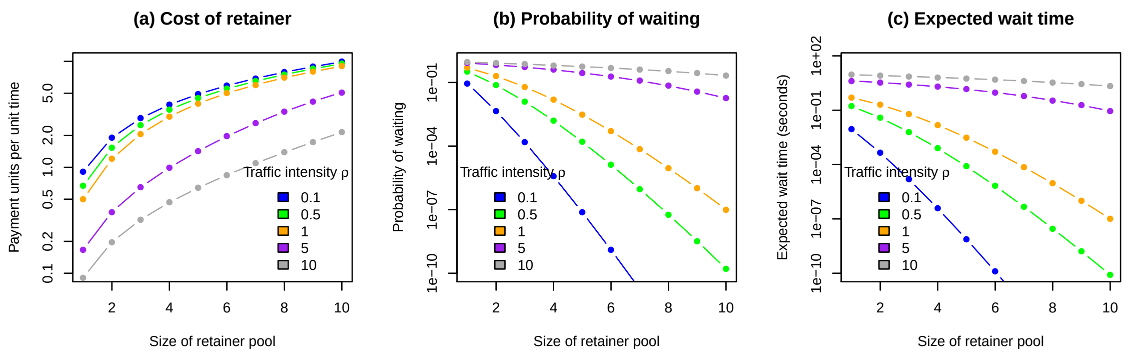


Figure 4-5: Graphs that visualize the relationships between retainer pool size, traffic intensity, and (a) cost, (b) probability of a task waiting, and (c) expected wait time. In the graph of expected wait time, we set $\lambda = 1$, so $\mu = \rho^{-1}$. When $\rho > c$, there are often not enough workers on retainer to service all tasks. As a result, wait time goes up, but cost goes down.

We can derive the closed form expression of the expected number of busy servers:

$$\begin{aligned} E[i] &= \frac{\sum_{i=0}^{c} i\rho^i/i!}{\sum_{i=0}^{c} \rho^i/i!} \\ &= \rho\frac{\sum_{i=0}^{c-1} \rho^i/i!}{\sum_{i=0}^{c} \rho^i/i!} \\ &= \rho(1-\pi(c)) \end{aligned} \tag{4.4}$$

In steady state, we need to pay all retainer workers who are *not* busy. That is, we expect to have $c - \rho(1-\pi(c))$ workers waiting on retainer. If our retainer salary rate is s (e.g., $s = \frac{1}{2}¢$), we would pay $s(c - \rho(1-\pi(c)))$ per unit time on average.

Visualizing the Relationships

While these equations give us precise relationships, they may not convey intuitions about the performance of the platform. Figure 4-5 plots these relationships for several possible values of ρ. The curves have a knee at $c \approx \rho$ for getting a good probability of response. A pool size $c > \rho$ means that an empty pool's *overall* rate of recruitment of workers, $c\mu$, exceeds the arrival rate of tasks. In other words, we begin to catch

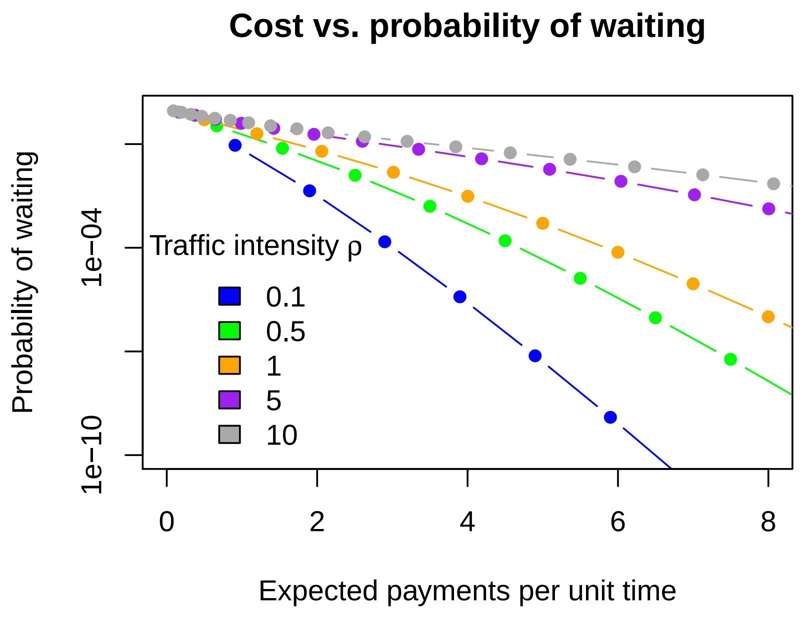


Figure 4-6: By calculating cost and the probability of a task needing to wait for integer values of $c \in [1, 15]$, we can visualize the relationship between the two values.

up and rebuild a set of available workers.[5] On the other hand, if $c < \rho$, then even an empty queue will not recruit workers fast enough to cover all arriving tasks, so it will stay empty.[6]

Figure 4-6 visualizes the relationship between the requester's cost and the probability of waiting. We derive this parametric curve by choosing values of c, then finding the cost and probability of waiting given that value. Paying more (i.e., for a larger pool) always improves the probability that the system can immediately handle a request. However, for small values of ρ, e.g. $\rho \leq 1$, paying 1–1.5¢ per minute brings the probability of waiting near zero. When tasks arrive quite quickly, 2.5¢ or more is necessary to achieve similar performance.

[5]When $\rho/c \to 0$, the number of free workers goes to $c - \rho(1 - \rho^c/c!)$, or effectively c.
[6]As $\rho \to \infty$, the number of free workers goes to $c - \rho c/(\rho + c) = c(1 - \rho/(\rho + c))$ which goes to 0.

4.2.2 Optimal Retainer Pool Size

A queueing theory model allows us to determine the number of workers to keep on active retainer. The size of the retainer pool is typically the only value that requesters can manipulate, and it impacts both cost and expected wait time. Requesters want to minimize their costs by keeping the retainer pool as small as possible while also maintaining a low probability that the task cannot be served in realtime. In this section, we present techniques for choosing the size of the retainer pool.

Our goal is to find an optimal value of c, given 1) the arrival rates λ and μ, and 2) desired performance, in terms of the probability of a miss $\pi(c)$ or total cost. We assume that the requester knows λ and μ either through empirical observation or estimation. We also assume that λ and μ are constant, but it is enough just for them not to change too quickly.

One approach to finding c is to specify the maximum allowable expected wait time for a task, or (equivalently) the maximum allowable probability that an incoming task will not be served in realtime. The intuition for this approach can be seen in Figure 4-5(b): if $\rho = .5$, for example, and the requester wants a less than 5% probability of any given task needing to wait, then $c = 3$ is the smallest retainer pool that can make such a guarantee.

Algorithmically, if $p_{\max}$ is the maximum desired probability of a task not being served in realtime, we want to minimize c subject to $\pi(c) \leq p_{\max}$. To find the solution, we use a binary search over possible values of c.

A more interesting version of the problem is for the requester to attach a dollar value to each task that cannot be serviced in realtime. For example, some pizza delivery companies do not charge the customer for the pizza if they cannot deliver it within thirty minutes. A miss then costs the company the value of the pizza plus the deliveryman's wage spent delivering the late pizza. A requester might similarly offer the service for free if it is not completed in realtime, or they might decide that the bad experience of a non-realtime result is worth \$1 in lost potential revenue from that user.

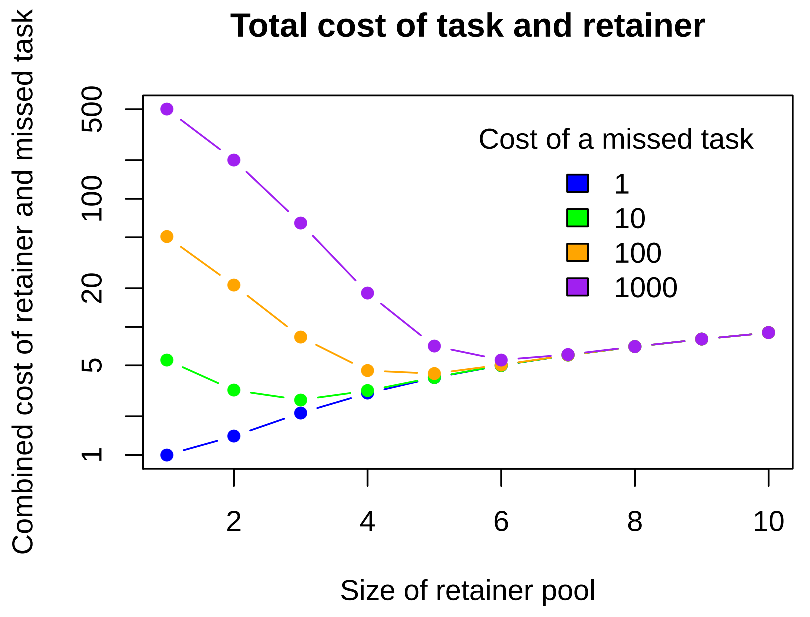


Figure 4-7: By assigning a dollar value to missed tasks, we can visualize the relationship between retainer size and total cost. Assuming traffic intensity $\rho = 1$ and retainer wage $s = 1$, these curves demonstrate the trade-off between more missed tasks on the left part of the graph and higher retainer costs on the right.

It now becomes possible to directly minimize the requester's total cost. Let C_{total} be the expected total cost to the requester and C_{task} be the loss if a task is not completed in realtime. Then C_{total} is the sum of the expected task cost—zero if addressed in realtime, or C_{task} otherwise—and the wage for the retainer workers derived from Equation 4.4:

$$C_{total} = C_{task}\pi(c) + s(c - \rho(1 - \pi(c))) \tag{4.5}$$

We can minimize this total value. Figure 4-7 shows this curve for several possible values of C_{task} when $\rho = 1$. The minimum value on the y axis for each curve is the optimal retainer size.

4.2.3 Worker Abandonment

Queueing theory models assume that a server will always begin a job once it is assigned. However, workers will sometimes leave the computer, close the window, or otherwise not respond to the retainer alert. Empirically, Section 4.1.2 found that about 10-20% of workers on active retainer never responded.

Our model can be adapted to capture worker abandonment. Let a be the percentage of workers who abandon the task, that is, they do not return after the retainer alert. A straightforward edit is to add the constant a to the probability that a task will not be serviced in realtime, so that probability is now $a + \pi(c)$. The response to this would be to increase c to cover the difference and recall $1/a$ workers for each job instead of 1. However, this is a conservative approach.

A more cost-effective approach would be to alert another worker if the first worker does not respond quickly. If the mean worker response time to an alert is R, choose a scalar α and wait until αR for the worker to respond. If the worker has not responded by then, the platform immediately alerts another worker and waits another αR seconds before issuing a third request. There is a constant probability of a worker responding within time αR, so the expected number of alerts before getting a response will likewise be a constant.

Unfortunately, queueing theory cannot easily accommodate this kind of approach. A model including server breakdown is a close match, except that server repair recruits another worker, which means that task arrivals are correlated and no longer Poisson. To bound the expected cost within the queueing theory framework, we envision a more complicated construction, which would be unlikely to be used in a running system.

In this construction, we maintain several tiered retainer pools and cascade down the tiers if we are having trouble finding a worker quickly. Specifically, if a worker in tier i does not respond within time αR, we alert a worker in tier $i + 1$. Each request has a known probability of succeeding in time αR, as reported in Figure 4-3. Task arrivals to each queue are now Poisson, since a constant fraction of the requests to tier i will pass through to $i + 1$. For example, we might choose α such that half of the requests will respond in time. Then, if tier i has task arrival rate λ, tier $i + 1$ would have arrival rate $\lambda/2$. We would see a sequence of geometrically decreasing pool sizes: for example, a top tier pool of 100 workers, then a second-tier backup of 50, a third-tier backup of 25, and so on. So, the total cost, a geometric sum, will be a small multiple of the first-tier cost, which we have already analyzed.

4.2.4 Limited Retainer Lifetimes

This model does not currently capture limited retainer lifetimes. In particular, the current implementation of retainers gives workers a lump sum for a task and a fixed retainer period. However, the model assumes that it can pay workers indefinitely to keep them on retainer. To capture fixed retainer timeouts, the system would need to model the probability that a worker reaches the end of the time period without being given a task. Each worker who times out incurs a fixed cost (equivalent to one task) and also leaves an empty spot in the retainer pool that needs to be replaced. The probability of an unused worker depends most directly on λ and c: more task arrivals mean workers wait less time, and the existence of more workers raises the time each worker must wait. In this situation, the platform could optimize the retainer timeout to balance low expected wait times vs. the additional cost of retainer timeouts.

4.3 Application to Common Crowdsourcing Tasks

Realtime crowds enable systems to execute common crowdsourcing tasks like votes very quickly. Rather than waiting minutes or hours, the system can now expect wait times on the order of seconds.

A simple example is A/B, a low-latency crowd voting platform. It can be hard to escape from our own biases when we try to predict what others will think. Crowds are certainly good at having opinions, but high latency makes them less useful for snap decisions.

A/B is our lowest-latency crowd feedback platform. Processes took upwards of

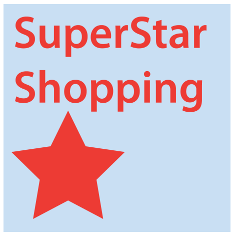
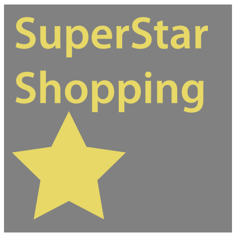
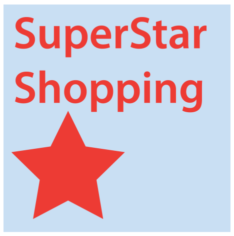
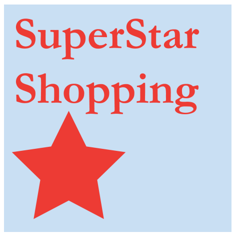
Question	Winner	Loser	Time to Five Votes
Which is scarier?			5.8 seconds
Which person looks more heroic?			8.8 seconds
Which logo looks better? [Color Version]			7.7 seconds
Which logo looks better? [Serif/Sans-Serif Version]			6.4 seconds

Table 4.3: A/B captures quick crowd votes.

twenty minutes for a Find-Fix-Verify workflow in Soylent and sixty seconds for a single vote in previous work [21]. A/B returns five votes in as little as five seconds. The user asks a question and takes two pictures, then a histogram of crowd feedback appears moments later. A user might try on two different sweaters, take pictures of each, and ask which one looks better; an artist might sketch two different versions of a character and ask which one looks more engaging; a designer might want fast aesthetic feedback on a sketch.

In our tests with eight workers on retainer, A/B returned five opinions in roughly five seconds (Table 4.3). Chapter 5 will extend these results to more complex systems.

4.4 Improvements to Crowdsourcing Platforms

So far, we have analyzed the retainer model as it could be implemented on top of existing crowdsourcing platforms. However, by extending the platforms, we can improve the recruitment performance considerably. This section combines some of the formalisms from the queueing theory model with platform changes and experiments to demonstrate the impact of the extensions.

4.4.1 Retainer Subscriptions

Just like the retainer model needs to expect a short wait after it alerts workers to return to the page, it also needs to wait $1/\mu$ seconds on average to recruit a new worker into the retainer pool. This recruitment time is a limiting factor of the retainer model. A small arrival rate means that the retainer pool can take a long time to fill, which is particularly problematic for large bursts or tasks that need multiple simultaneous workers.

One way to increase μ is for the platform to put together a panel of *retainer subscribers* who can be directly notified when the retainer pool needs to recruit a replacement. Imagine, for example, sending an instant message to subscribers when a position opens up in the retainer pool. The insight behind this approach is to change from a *pull* model of crowdsourcing, where workers seek out tasks, to a *push* model, where tasks offer themselves to workers. Workers could subscribe to a *task type*, so that when the platform needs a retainer worker for a task of that type, the platform could send a dialog notification to one or more subscribed workers and offer them the opportunity to complete one task in the next few minutes. Workers who accept are now on retainer, can continue working on other tasks, and will be interrupted whenever the realtime task arrives.

A push notification is likely to reduce the time it takes to recruit a worker onto retainer, thereby increasing μ.

4.4.2 Global Retainer Pools

In the previous analysis, each requester maintained their own retainer pool. In this section, we analyze how sharing one global retainer pool across requesters improves performance. We also investigate how to route tasks to workers in a globally pooled retainer.

Global Pool Analysis

In this section, we turn to the queueing theory model to understand how combining retainer pools will impact $\pi(c)$, the probability of a missed task.

Another way of writing $\pi(c)$ in Equation 4.1 is $\pi(c) = \pi(0) \cdot \rho^c/c!$, where $\pi(0) = (\sum_{i=0}^{c} \rho^i/i!)^{-1}$ [69]. To make this equation easier to manage, recall Stirling's approximation that $c! \approx \sqrt{2\pi c}(c/e)^c$. Also note that the sum that defines $\pi(0)$ is decreasing geometrically, so we can approximate $\pi(0) \approx e^{-\rho}$, a constant. This approximation gives us:

$$\pi(c) \approx \sqrt{2\pi c}\left(e^{-\rho}(e\rho/c)^c\right) \tag{4.6}$$

If we have k different tasks each with traffic intensity ρ and queue size c, the probability of each empty pool is $\pi(c)$. Each requester independently suffers, so the total probability across all requesters is multiplied by a factor of k. So, the probability of an empty pool existing somewhere among k requesters each with c retainer workers:

$$\text{k independent } \pi(c) \approx k\sqrt{2\pi c}\left(e^{-\rho}(e\rho/c)^c\right) \tag{4.7}$$

Now suppose we bring all the retainer pools together, creating one "superpool" of size kc. The task arrival rate λ increases by k but the rate at which we recruit one worker μ remains unchanged. Thus the traffic intensity increases by a factor of k to $k\rho$. So, the probability of an empty pool with combined retainers is

$$\text{k combined } \pi(c) \approx e^{-k\rho}\sqrt{2\pi kc}(e\rho/c)^{kc} \tag{4.8}$$

$$= \sqrt{2\pi kc}\left(e^{-\rho}(e\rho/c)^c\right)^k \tag{4.9}$$

Ignoring the square root factor, we see the main term being *exponentiated* by a factor of k. In other words, the loss rate declines exponentially with the number of retainer pools we bundle.

We can look at some approximations for these results. We can investigate $\pi(c)$, the probability of having all servers busy. Suppose we set $c = (1+\epsilon)\rho$, just above our $c \approx \rho$ knee in the curves from Figure 4-5. Then, with a single retainer pool, $\pi(c)$ is about

$$\begin{aligned} e^{-\rho}\sqrt{2\pi c}(e\rho/c)^c &\approx e^{-\rho}(e/(1+\epsilon))^{(1+\epsilon)\rho} \\ &= e^{\epsilon\rho}/(1+\epsilon)^{(1+\epsilon)\rho} \\ &= \left(\frac{e^{\epsilon}}{(1+\epsilon)^{1+\epsilon}}\right)^{\rho} \end{aligned} \tag{4.10}$$

This is the same quantity as shows up in the typical analysis of the upper tail of the Chernoff bound. There, we generally approximate this quantity as $e^{-\epsilon^2\rho/3}$, which is reasonably accurate for any $\epsilon < 1$. In short, the probability of an empty pool is roughly $e^{-\epsilon^2\rho/3}$.

Using this approximation, we can ask how small a retainer pool is necessary to match the same $\pi(c) \approx e^{-\epsilon^2\rho/3}$ in the *globally shared* case as we found in the singular case. As we argued above, moving to the globally shared case multiplies ρ by a factor of k because the task arrivals are more frequent. Since the exponent we care about is proportional to $\epsilon^2\rho/3$, we can decrease ϵ by a factor of $\sqrt{k}$ and end up with the same bound as the singular case. In other words, the fraction ϵ of "buffer" workers that we need in our retainer pool is proportional to $\sqrt{k}$, as compared to the factor k in the singular case. We thus need many fewer extra workers per extra task: much like standard error decreases by a square root factor as sample size increases, we have less uncertainty in arrival rates as more requesters join together.

Task Routing

Shared retainer pools introduce speed and cost improvements, but workers will subscribe to multiple realtime task types and can only work on one realtime task at a time. This situation immediately raises the question of how to decide which worker should be assigned to each retainer pool when a spot opens up. Market forces like task pricing will help solve this problem, but microtask markets like Mechanical Turk are clustered on a small number of prices (often $2 - 5$¢). Inefficient task routing could lead to logjams where certain tasks cannot find workers. In this section, we demonstrate that a straightforward approach like uniform randomization could lead to extremely slow response times, and we introduce a linear programming solution that optimizes response times across tasks.

Suppose we have a set of task types $T = t_1, \ldots, t_n$, and tasks of type t_j arrive with Poisson distribution and rate λ_j. Not every worker can complete every task: workers may have only signed up to be on retainer for particular task types, or they may not have the qualifications for all task types. We split workers into groups $w_1, \ldots, w_m$ that are uniquely identified by the tasks that group can complete. So, for example, w_1 might represent all the workers who can complete tasks t_1, t_2, and t_3. We say that W is the set of all worker types $W = \{w_1, \ldots, w_m\}$, and that each w_i has a Poisson arrival rate μ_i.

Given a set of task types T, a set of worker types W, and arrival rates for each, our goal is to assign workers to tasks to maximize the throughput of the system. To do so in steady state, we need to decide how many worker arrivals—more precisely, what portion of the overall arrival rate—from each group should be assigned to each task. Let us say that the rate at which workers from group w_i should be assigned to tasks of type t_j is a_{ij}. These assignments must sum to the total arrival rate of the worker group: $\sum_{j=1}^{m} a_{ij} = \mu_i$. For example, in our earlier example of w_1, if $\mu_1 = 1$, one possible assignment is $a_{11} = .5, a_{12} = .25, a_{13} = .25$.

A standard approach would be to assign each worker arrival randomly to one of the task types that he or she can complete. (That is, a_{ij} are equal for any i.) However, this approach could result in slow completion times. In Figure 4-8, w_3 has four times the arrival rate of w_1 or w_2. Random assignment would send workers to t_3 at rate $1/4 + 1 = 5/4$, whereas t_1 would receive workers at just $1/2$. Depending on which workers are online, each of the task types could find itself in a similarly starved state.

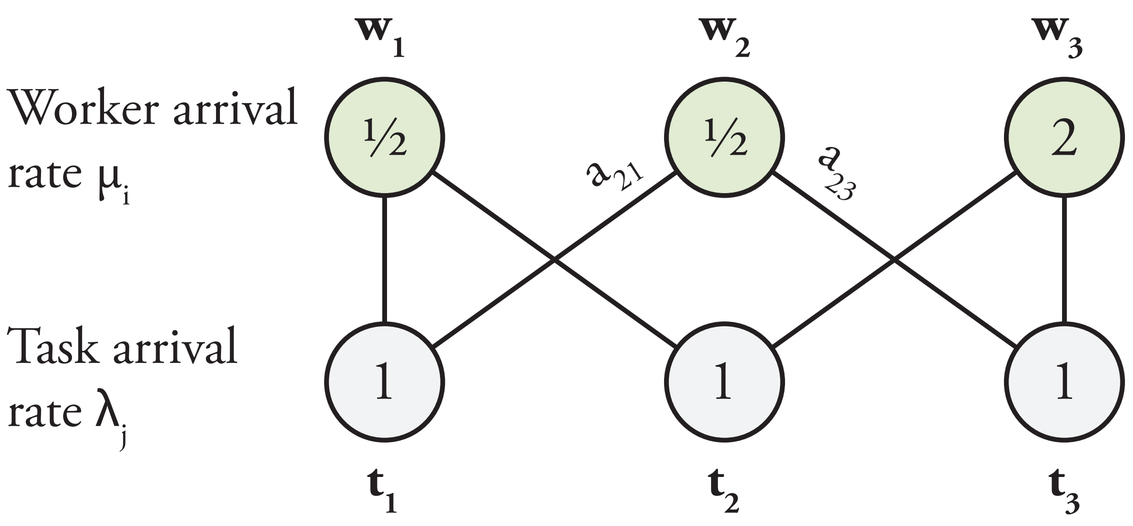


Figure 4-8: A task routing scenario where a typical randomized approach would lead to poor results. t_1 would receive relatively few workers. Depending on the values of μ_i, each task type could find itself in this starved state.

Instead, a centralized system can route workers to equalize traffic intensity across all tasks. This goal can be described as a linear programming problem, but in fact can be solved using maximum flow, which is significantly faster than general linear programming. The following constraints suffice to define a linear programming problem — they indicate that the incoming worker rate to each task type is at least as high as the incoming task rate, and that all the worker assignments from a worker group sum to no more than the arrival rate of that worker group.

$$\begin{aligned} \sum_i a_{ij} &\geq \lambda_j \text{ for all } j, \\ \sum_j a_{ij} &\leq \mu_i \text{ for all } i. \end{aligned} \tag{4.11}$$

These constraints will yield a solution, but that solution may not accomplish the requesters' goals. So, we can instead choose to be more specific about the quantity to maximize. For example, as we have seen above, task wait times are typically a function of the ratio of arrival rate and service rate (λ/μ), known as traffic intensity ρ. We can define an analogous ρ here to be the ratio of the incoming task arrivals to the summed rate of arrivals from all worker groups for that task: $\rho = \lambda_j / \sum_i a_{ij}$. We then minimize its worst case across all tasks:

$$\begin{aligned}
&\text{minimize } \rho \\
&\text{subject to} \quad \rho \sum_i a_{ij} \geq \lambda_j \text{ for all } j, \\
&\qquad\qquad\quad \sum_j a_{ij} \leq \mu_i \text{ for all } i.
\end{aligned} \tag{4.12}$$

Minimizing ρ across all tasks guarantees that all tasks receive workers in similar proportion to their task arrival rate. So, it would be rarer to see one task flush with workers while another one waits.

By merging retainer pools, the platform can thus help guarantee fast results for all tasks.

Scaling

One practical difficulty with this approach is estimating μ_i as the number of task types grows. If there are $|T|$ different task types, there are $2^{|T|}$ different combinations of task types that a worker can sign up for, and thus $|W| = 2^{|T|}$. This set is an extremely large number of arrival rates to try and estimate accurately, and will make the linear program hard to solve because there will be an exponential number of constraints.

However, the problem of efficient feature representation is a common one in machine learning. There are many approaches to this problem. We may find that in practice only a small number of task type combinations can occur. We can also enforce this, for example by setting a ceiling on the number of task types a worker can subscribe to at once. With a limit of two subscriptions, $|W| = |T|^2$ instead of $2^{|T|}$.

4.4.3 Precruitment: Predictive Recruitment

So far, we have been limited by the length of time it took a worker to respond to the retainer alert. However, our model suggests that even this two-second barrier (Section 4.1) is unnecessary, and that crowds could be recruited effectively instantaneously.

The insight behind our solution is *precruitment*: notifying retainer workers before the task actually arrives. The queueing theory model involves estimating $1/\lambda$, the expected length of time before the next task will arrive. If $1/\lambda$ is about the length of time it takes to recall a retainer worker, we can recall a retainer worker and expect to have a task by the time the worker arrives. As we will demonstrate, workers are also happy to wait at a "Loading..." screen even if the task is not ready immediately.

Workers take 2-3 seconds to arrive (Section 4.1) and will wait for roughly ten seconds afterwards [150]. The Poisson task arrival process has rate λ, and Poisson distributions have standard deviation $\sqrt{\lambda}$. So, the platform can precruit $\lambda + \beta\sqrt{\lambda}$ workers per second for upcoming requests, where β is a slack variable that controls how many extra standard deviations to precruit for safety. Any workers who do not have tasks within a predetermined wait time would need to be paid and dismissed. However, as the platform becomes large and λ grows, the standard deviation will become proportionally smaller relative to the mean, making it possible to waste very little money on extra workers.

In fact, the entire precruitment system can be represented as its own M/M/c/c queueing system. Many of the same techniques introduced earlier can be applied to help optimize the size of a precruitment pool in relation to the standard retainer pool.

4.4.4 Evaluation

We ran a study on Mechanical Turk as a proof-of-concept for precruitment. In the study, we followed the protocol of Section 4.1.2 by offering three cents for a one-minute retainer task: a game of Whack-a-Mole (Figure 4-9). After waiting on retainer for one minute, workers responded to the retainer alert and were asked to quickly click on the picture of a mole randomly placed in a 3x3 grid of dirt mounds. However, after responding to the alert and before the mole appeared, workers needed to wait for a randomly selected length of time between 0 and 20 seconds while a "Loading..." indicator displayed.

We measured the length of time between the appearance of the mole and: a) mouse

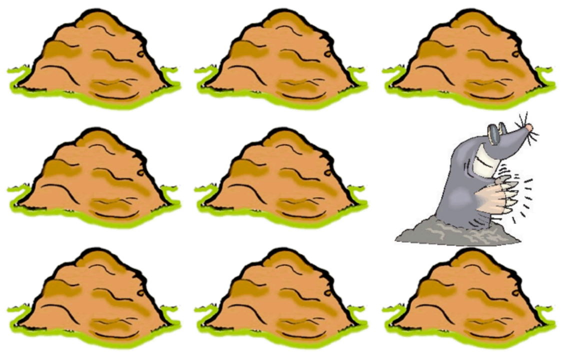


Figure 4-9: To test precruitment, workers participated in retainer tasks that challenged them to play a game of Whack-A-Mole and click on the mole as soon as it appeared.

movement in the direction of the mole, and b) the click on the mole. We discarded a small number of responses where the worker clicked on a dirt mound instead of the mole or where the browser did not record millisecond-precision timing. After filtering, our dataset consisted of fifty workers who completed N=373 Whack-a-Mole tasks. One limitation of our design is that Whack-a-Mole is a relatively enjoyable task, and workers might not be so attentive for less game-like tasks.

The median length of time between the mole's appearance and the worker moving the mouse toward the mole to click on it was 0.50 seconds across all wait times (mean 0.86, std. dev. 1.45, Figure 4-10). The median length of time before clicking on the mole was 1.12 seconds (mean 1.87, std. dev. 2.23, Figure 4-11). There is a negligible correlation between wait time and mouse movement delay ($R^2 = .001$), suggesting that workers react roughly as quickly right after they arrive as they do twenty seconds later.

We can use the same dataset to compare precruitment to the retainer approach without precruitment. This comparison is possible because a "Loading..." delay of zero seconds is the exact same worker experience as the standard retainer model. We are interested in the lag time between the task arriving and mouse movement to whack the mole. Here, a new task results in an alert being sent to the worker, so we start our timer with the alert. Without precruitment, the median time between task posting and mouse move was 1.36 seconds (mean 1.41, std. dev. 0.30).

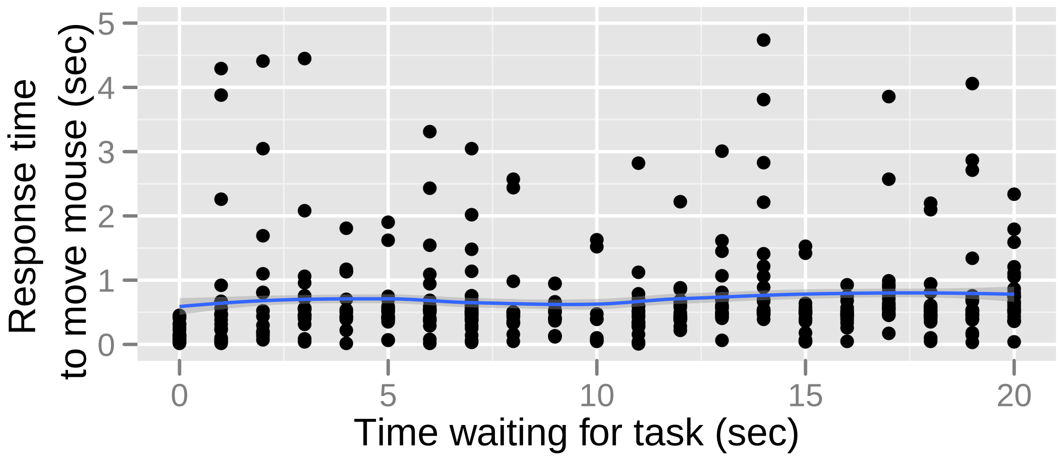


Figure 4-10: The median length of time between the mole image appearing and the workers moving to click on it was 0.50 seconds. So, a platform can recall retainer workers early and get crowds in half a second instead of waiting for the workers to respond to the retainer alert.

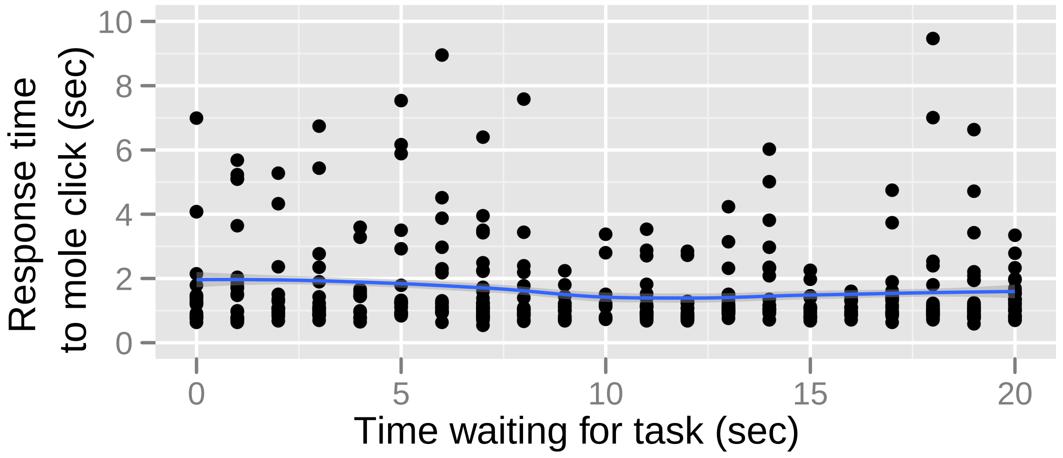


Figure 4-11: The median length of time between the mole image appearing and the workers clicking on it was 1.12 seconds.

This result suggests that, had we used a standard retainer model with this task, we would have seen mouse movement typically after 1.36 seconds. Using precruitment, we get mouse movement in 0.5 seconds. Precruitment finally breaks through the sub-second cognitive barrier that keeps users in flow [150].

4.5 Discussion

Our model has several limitations. One limitation is that an M/M/c model may be a better match for certain retainer implementations where the crowd is of limited size and the system must handle tasks FIFO instead of recruiting an additional crowd member when the retainer pool is empty. Second, worker recall delays depend on the length of time the worker has been waiting on retainer (Section 4.1), but our analysis ignores this fact. Third, our model assumes that it can always recruit new retainer workers into the pool, but the retainer population is limited in practice. However, we believe that these observations can be integrated into our optimizations.

One empirical question we have not addressed is the number of workers that need to be on a crowdsourcing platform to make sure that requesters can maintain full retainer pools. This number also depends on the percentage of workers who are willing to sign up for retainer tasks. Since the retainer model pays more than batch tasks, we anticipate that this percentage will be high. On Mechanical Turk, our experience is that it is not difficult for a single requester to get twenty or thirty workers on retainer simultaneously. However, as more requesters use retainers, these dynamics may shift.

While realtime retainers are the motivating example here, the entire Mechanical Turk platform can be thought of as a large retainer system where workers are paid zero retainer wage and the worker recall rate is extremely slow, since workers return on their own initiative rather than by recall. Precruitment is another kind of retainer model queue where workers are recalled before the task even arrives. All three queues could be analyzed together as a queueing network in order to more effectively understand the entire system. However, it is also possible to bound the probability of a slow task response via the probability that any of the retainer pools are empty. Supported by our results so far, we suggest that queueing theory can be applied for many other problems in the space of realtime crowdsourcing as well.

Our analysis suggests that paid crowdsourcing platforms could integrate a globally-managed retainer into their design. This will not only change the types of crowdsourcing that are common, but will also introduce new elements of worker reputation. We suggest two new reputation statistics. First, a worker's median response time characterizes how quickly they respond to the alert and begin working on a retainer task. Requesters prefer workers with low response times. Second, workers are tagged with a response rate: the percentage of the time that they successfully respond to a retainer alert. If a worker does not respond to the alert within a given length of time (e.g., five seconds), the system finds another person and the worker is not paid.

Chapter 5 will extend these ideas into realtime crowd-powered systems. In these systems, minimizing recruitment time will not be enough. The challenges of realtime results in the face of slow *work time* will inform a new set of techniques for synchronous crowds.

Chapter 5

Realtime Crowdsourcing: Systems

The retainer model opens the door to create realtime crowd-powered systems. The previous chapter demonstrated how realtime crowds could be tasked with traditional crowdsourcing tasks and complete them in seconds. This chapter opens the design space of much more complex systems, as well as algorithms to coordinate realtime crowds.[1]

The core insight behind complex realtime crowd-powered systems is the use of on-demand *synchronous crowds*. In synchronous crowds, all crowd members arrive and work simultaneously. This simultaneous crowd work enables realtime coordination and collaboration to complete tasks much faster than parallel, asynchronous workers. The retainer model is the first approach to guarantee synchronous crowds, and we make use of them here.

Developers need ways to guide or program synchronous crowds for realtime results. We introduce *rapid refinement*, the first algorithm for synchronous crowds. Rapid refinement focuses on low-latency, reliable work. The fundamental insight behind rapid refinement is that *synchronous (simultaneous) crowd work enables an algorithm to recognize agreement early.* The rapid refinement design pattern quickly reduces

[1]This chapter has adapted, updated, and rewritten content from a paper at UIST 2011 [16].

a large search space by focusing workers' attention to areas they are beginning to agree on independently (Figure 5-2). Repeatedly narrowing the search space to an agreement region encourages quality results because it is built around independent agreement. It also allows the interface to give the user incremental, trustable feedback before a final answer is available.

We use the retainer model and rapid refinement to explore new avenues for realtime crowd-powered interfaces through a system called *Adrenaline*. Adrenaline is a smart camera shutter powered by crowd intelligence: it finds the right moment to take a photo. Instead of taking a single shot, Adrenaline captures a short video — allowing the photographer to move around the scene, the subject to strike multiple poses, or action in the scene to unfold unpredictably — then uses rapid refinement to identify the best moment as a still photo about ten seconds after the shutter closes. Low latency means that users can preview and share photos they just took, like they would with any other digital camera.

This chapter introduces 1) the rapid refinement technique for fast crowd search through a continuous space, and 2) the Adrenaline camera. It then reports on an evaluation which verifies that rapid refinement leads to fast results: faster than approaches that keep workers separate, and faster on average than even the fastest individual worker. Finally, we explore how realtime crowd-powered systems can support creative applications.

5.1 Adrenaline

> *"You must know with intuition when to click the camera. That is the moment the photographer is creative. [...] The Moment! Once you miss it, it is gone forever."* – Henri Cartier-Bresson [12], 1957

Photographers often struggle to capture what Cartier-Bresson called The Decisive Moment [12]. The instant when the shutter opens, the subject might have broken into an awkward smile, the angle might be poor, or the decisive moment might have passed. As a result, photos taken by novices can feel stilted or 'off'. Experienced

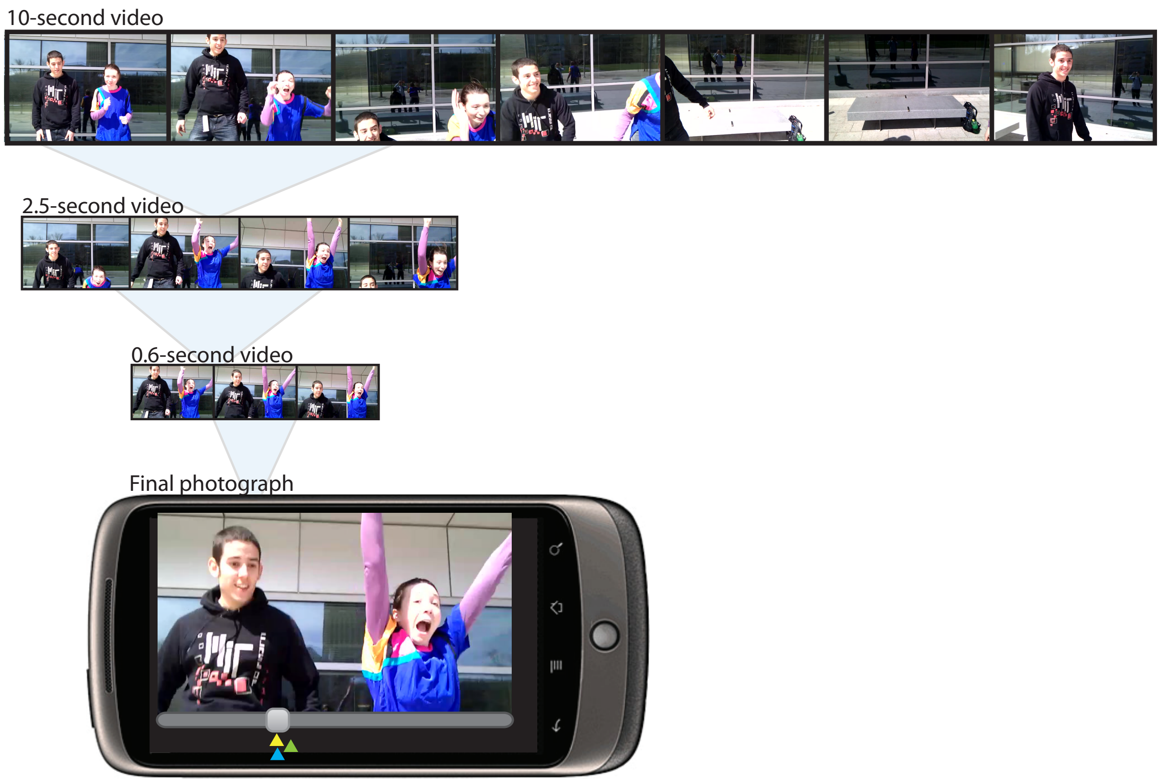


Figure 5-1: Adrenaline is a camera that uses crowds to find the right moment for a picture by capturing ten-second movies. It looks for early agreement to filter the timeline down quickly to a single frame. A photo is typically ready about one second after the user reviews the movie.

photographers learn to compensate by taking many photographs — tens of photos rather than one — then sorting through them later to find the gem. They try multiple angles, capture photos over several seconds, or ask subjects to strike different poses.

Adrenaline is a realtime crowd-powered camera that aims to find the most photogenic moment by capturing many alternatives. Rather than taking a single photo, Adrenaline captures a ten-second movie and recruits a crowd to quickly find the best photographic moment in the movie (Figure 5-1). Within five seconds after the movie is captured, the user can see crowd members exploring the timeline (colored triangles in Figure 5-1). A few seconds later, the user can see that the crowd has narrowed down to a small fraction of the timeline, then again to a few adjacent frames, and finally to a final photo in a total of about eleven seconds. This means that a final photo is ready just moments after the user finishes reviewing the movie they just captured.

Adrenaline's goal is to mimic the instant-review capabilities of cameras today. Seeing a photo quickly means that users can take another photo if they want, show it instantly to friends, or post it to the web. The visceral thrill of novice photography can be lost when the photo is not available for minutes or days. Many novices never review the pile of extra photos, so we believe that is extremely important to complete the image selection process while Adrenaline still has the user's attention.

Existing crowdsourcing techniques cannot support Adrenaline's goal of 10–12 second latency. Using the retainer model, crowds arrive quickly, but it takes them a long time actually choose the best frame. We thus introduce the *rapid refinement* algorithm, which guides the search process quickly and reliably.

5.2 Rapid Refinement: Coordinating Synchronous Crowds for Fast Results

Once recruitment times are negligible, slow work time dominates the user experience. The problem is not just minimizing average response time, but also minimizing variance. Time variance means that wait time will not be reliable: it will depend on whether the system happened to recruit a fast worker. That worker may also produce low quality results. To solve this, human computation algorithms such as Find-Fix-Verify often wait for several results before proceeding: it is even less likely that a system would recruit multiple fast workers. We could require workers to finish within a short time limit, but our experience is that this is stressful to workers and leads to poor results.

To complete nontrivial crowdsourcing tasks in realtime, we must develop new algorithms and programming patterns to return quality results quickly and reliably. Like traditional randomized algorithms, we may be willing to sacrifice correctness guarantees in exchange for faster runtime.

The insight behind our solution is that *low-latency crowds are synchronous crowds*:

all workers are working simultaneously. Synchronous crowds can interact with, influence, and communicate with each other and the user. In most on-demand crowdsourcing approaches, workers do not overlap in time enough for synchronous designs to make sense. However, the retainer model makes it practical to assume, for the first time, that the crowd is all present simultaneously a few moments after request. So, rather than waiting for all results in order to continue, a realtime algorithm has the opportunity to influence the process *as it takes place.*

We take advantage of synchronous crowds by recognizing potential agreement early on, while workers are still exploring, and then focusing workers' attention on the area of agreement. The insight is that the majority of Turkers who are efficient satisficers [177] can guide the process: these workers decide on the gist of the solution quickly, but can take time to commit to a final answer. Rapid refinement recognizes when several workers are likely to agree and commits for them by focusing the task on that area. Fast workers will still find an answer quickly and contribute to the vote, whereas slow workers benefit from the focus on a smaller search space. In the next section, we will describe the rapid refinement algorithm that implements this idea.

5.2.1 Algorithm Design

The rapid refinement algorithm repeatedly narrows down a search region to a fraction of its existing size when it senses that workers independently agree on the subregion (Figure 5-2). It is appropriate for human computation tasks where the workers' quality or utility function is continuous. Other examples in the photography domain include brightness, contrast, color curves, and zoom level.

Rapid refinement begins with the entire search space available and workers initialized to a random position. The algorithm takes place in *phases* where each phase narrows to a smaller search region. Workers trigger a new phase by independently focusing on the same area for a period of time. The algorithm depends on three values: the agreement range r, agreement time t, and agreement amount a. If a fraction a of the workers stayed within a range less than a fraction r of the current search area for at least t seconds, rapid refinement declares agreement. It then shrinks the search

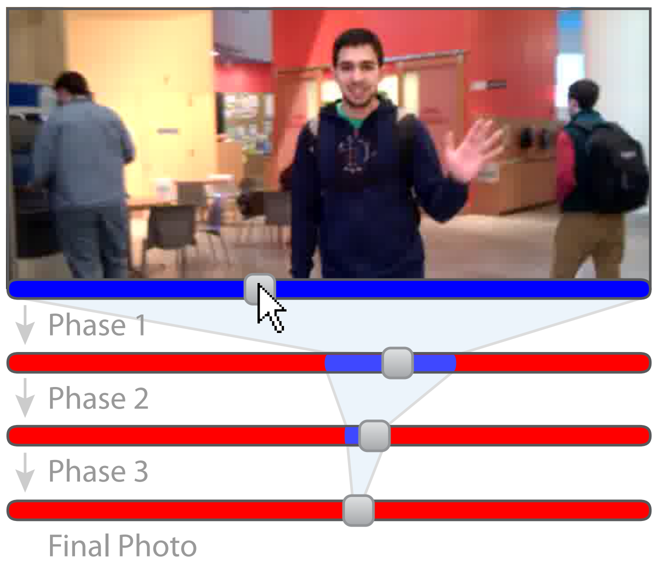


Figure 5-2: Rapid refinement repeatedly shrinks the working area for all workers when it detects that several independent workers are exploring the same area.

space to match the agreement range and begins a new phase. Workers are no longer able to explore the out-of-bounds area, and must try to agree within the new region. This process repeats until convergence. To approve or reject work, rapid refinement looks at whether the worker agreed with at least one phase change.

In Adrenaline, the algorithm begins with the entire video timeline. Workers vote on a good photo using their timeline slider. They cannot see others' sliders, which encourages independent agreement [187]. When at least $a = 33\%$ of the workers have been in the same $r = 25\%$ of the timeline for at least $t = 2$ seconds, Adrenaline declares agreement. These values can be adjusted to trade off delay for false positives. With these values, Adrenaline converges in 3–4 phases per ten-second input video. The first phase is the slowest, and agreement accelerates as workers' attention is focused on one area.

The rapid refinement algorithm has several benefits aside from speed. First, it produces preliminary results that can be returned to the user early. Early results reduce interface latency and allow the user to provide feedback on the process. For example, a future version of Adrenaline could play the refinement region to the user as soon as the workers converge on the first phase, and allow the user to adjust it if desired. Second, it combines work and verification into one stage, which saves cost and time for a separate verification step. Third, workers tailor their votes toward what someone else might think, which minimizes individual bias [72].

Rapid refinement makes tradeoffs. First, the algorithm may focus too early on a single part of the search space, especially if the low quality workers are the first to respond. With four or more workers, it may be possible to fork the crowd into two simultaneous groups to help avoid this problem. Forking the crowd has the additional benefit that the algorithm can explore multiple promising paths at the same time. The system can also watch for thrashing or otherwise confused behavior to detect if it has selected a poor subset of the video, then back off. Second, it may stifle individual expression: talented workers might be forced to agree with the majority. A future system could recognize such workers and give them more weight in the votes.

The algorithm can stall in theory if all workers stay far away from each other, but a slight modification can guarantee convergence. In particular, systems can start with a small agreement fraction r and increase it as time passes in each phase. Then, given at least three workers, the algorithm will always converge. It may also make sense to dynamically adapt a, r and t based on the phase or on worker behavior. However, in our experiments, rapid refinement never stalled.

5.3 Evaluation

We have argued that the retainer model and rapid refinement combine in Adrenaline to produce a realtime crowd-powered interface by controlling variance in wait time and quality. In this section, we report on an evaluation of Adrenaline that checked this claim, stress-tested the retainer model and rapid refinement, and investigated whether end users could use the system to take good photographs.

5.3.1 Method

We recruited 24 participants through e-mail lists in exchange for a $20 gift certificate. Fourteen were male and ten were female, and the median age was 25. About half had taken a photograph on a cell phone camera or consumer camera in the past month, and five had taken pictures on a DSLR camera in the past month. The typical participant was a young, technically competent student who had a moderate interest in photography as a hobby (median Likert response: 4/7). Participants arrived to the study in pairs.

We gave each participant a smartphone with Adrenaline installed and introduced them to the application. We did not allow participants to immediately see the still photos that Adrenaline produced, so that we could compare rapid refinement to other (slower) approaches. Participants began by taking a video portrait of their partner. Then, the pair spent fifteen minutes in a public area capturing videos of people, actions, or landscapes. Finally, participants chose two videos in addition to the portrait to submit for the evaluation.

We generated candidate photographs using the following computational, expert, and crowdsourced approaches:

- *Rapid Refinement:* we required five workers to be on retainer before labeling each video.
- *Generate-and-Vote:* a standard crowdsourcing approach [128] in two stages. First, five retainer workers independently selected a frame in the video. Then, keeping the fastest three results returned, we use eight retainer workers to vote on the best photo of the three.
- *Generate-One:* using the same dataset as Generate-and-Vote, this condition simulated stopping the process as soon as the first worker submitted a photo.
- *Photographer:* a professional photographer labeled the best still frame in each video.
- *Computer Vision:* the still frame selection algorithm on YouTube, which uses recent computer vision research algorithms [93].

We used the quikTurKit technique (Section 2.2, [21]) to repeatedly post tasks and keep them near the top of the Mechanical Turk task list, then implemented a five-minute retainer and a 2¢ bonus for quick response. We paid 4.5¢ on average for a rapid refinement or generate task (quikTurKit posts tasks at multiple price points simultaneously), and 3.5¢ on average for a vote task. Including bonuses and removing one worker on average who never responded to the retainer, these costs added up to 4(4.5¢) base + 2(2¢) bonus = 22¢ per video for the rapid refinement and generate tasks. Voting added eight workers: we estimated 7 would appear and 3 would earn the bonus, for an additional 31¢. So, rapid refinement and Generate-One cost 22¢ per video, and generate-and-vote cost 53¢ per video. In a live system, it would be feasible to use fewer workers and pay closer to 10¢ rather than 22¢. On Mechanical Turk, we posted half of the videos first in the rapid refinement and generate-and-vote conditions to compensate for order effects. All Mechanical Turk tasks were posted on a weekend afternoon.

We then contacted all of our participants and asked them to rate each still photo on a 9-point Likert scale. We instructed participants to ignore aspects of the picture like contrast and color that could be easily fixed in post-processing.

5.3.2 Results

We used the retainer model to post each video as soon as there were five workers on retainer, stress-testing the volume of the retainer model. Adrenaline had enough workers to label a video every 45 seconds (median). Worker arrivals were bursty, but the median time between retainer arrivals was 6.3 seconds. The median time between unique worker arrivals was 48.8 seconds. These numbers are dependent on the current workforce state of Mechanical Turk, and will change as the market grows or more tasks use retainers.

Timing

Table 5.1 and Figure 5-3 present the results. Rapid refinement returned the fastest results, with a median total time of 11.6 seconds ($\mu = 12.6$, $\sigma = 4.3$). Generate-One, which used the first available photo, was a few seconds slower. Its median time was

	Delay	
	Median	Mean
Generate and Vote	41.9	45.3 $\sigma = 14.0$
Generate One	13.6	16.3 $\sigma = 9.8$
Rapid Refinement	11.6	12.6 $\sigma = 4.3$

Table 5.1: Rapid Refinement was the fastest algorithm and had the lowest timing variance.

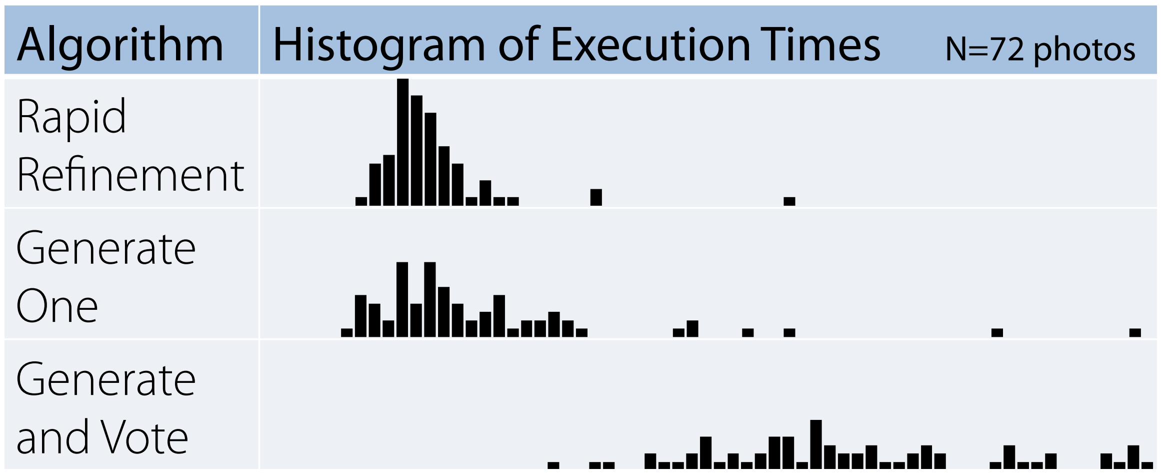


Figure 5-3: Rapid refinement consistently completed quickly. Generate One sometimes located a fast worker, but often did not, so it has a larger timing variance.

13.6 seconds ($\mu = 16.3$, $\sigma = 9.8$). Generate-One's timing standard deviation was nearly twice that of rapid refinement. These variances are significantly different from each other, F(71, 71)=5.17, $p < 0.001$.

Rapid refinement is faster than Generate-One especially because it accelerates the final stages of the selection. Single workers in Generate-One tended to identify the general region quickly, but spent time shuttling between a few frames before making a final decision. Rapid refinement instead encourages all workers to vote quickly and not worry too much about the final decision. Rapid refinement creates social loafing so that workers satisfice. When workers worry less about the final selection, they act faster and their collective wisdom leads to a good final selection.

The timing data is non-normal, so we square-root transformed it to satisfy normality assumptions. An ANOVA comparing the delay for the three human computation algorithms is significant F(2, 213) = 278.11, $p < .001$, and post-hoc pairwise Tukey comparisons confirmed that all conditions were significantly different than each other

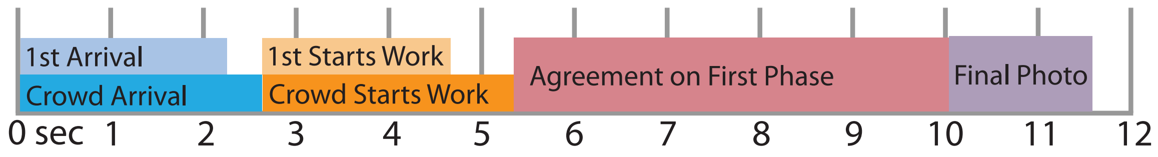


Figure 5-4: Timeline of the median Adrenaline execution. Each bar length represents the median length of time for that stage. For example, enough crowd members typically arrived 2.6 seconds after the request was made, and agreement on the first refinement typically took 4.7 seconds.

(all $p < .05$), confirming that rapid refinement is fastest.

Figure 5-4 outlines the median timing distributions of a rapid refinement process. One worker typically arrived 2.2 seconds after the video was uploaded, and the second and third came within 2.6 seconds. At least two workers were moving their sliders by 5.3 seconds after request. After the crowd began exploring, agreement took 4.7 seconds in the first phase. The median first phase completed a total of 10.05 seconds after request ($\mu = 10.65$, $\sigma = 3.0$). The following phases lasted 0.75 seconds each, typically. These phases moved more quickly because workers were often already agreeing on a small area when the first phase completed.

Quality

Experimental results validated our expectation that rapid refinement would sacrifice some amount of quality in exchange for faster results. However, on average it still produces high-quality photographs (Figure 5-5, Figure 5-6). Participants' ratings had high variance, making it difficult to draw statistical conclusions (Table 5.2). However, Rapid Refinement produces higher-quality photos than Computer Vision, which suggests that using crowds in a subjective photo quality task is a good match for human abilities. Due to the large variance, it is difficult to distinguish Rapid Refinement on average from Generate One and Photographer. As was the case with delay, however, Generate-One was less reliable and had a higher variance. Surprisingly, Generate-and-Vote appears to match the Photographer condition. While we believe that a photographer would take better pictures given the opportunity to operate the camera, it appears that an unskilled crowd may be equally talented at selecting good

	Quality (9pt Likert)
Computer Vision	4.9 $\sigma = 2.2$
Professional Photographer	6.4 $\sigma = 2.3$
Rapid Refinement	5.8 $\sigma = 2.2$
Generate One	5.9 $\sigma = 2.6$
Generate and Vote	6.6 $\sigma = 2.1$

Table 5.2: Rapid Refinement produced higher-quality photos than Computer Vision, faster than Generate and Vote, and with less variance than Generate One.

moments.

5.4 Realtime Crowd-Powered Creativity

In this section, we expand the design space of realtime crowd-powered interfaces beyond Adrenaline. Realtime crowds can support creative tasks in user interfaces as well.

Puppeteer (Figure 5-7) works in conjunction with Adobe Photoshop to support large-scale content generation and synchronous feedback to workers. Artists often want to create a large number of believably varying objects, like an excited audience at a concert or flowers in a field. Algorithmic techniques to generate poses may not be realistic, semantically meaningful, or generalizable to new objects.

Puppeteer users specify control points on an image using Photoshop's Puppet Warp tool [87] (Figure 5-7a). Users give a textual description of their goal (e.g., "Make the person look like he is dancing"), then workers each pose three figures to match the goal. As workers progress, the user observes them through a small multiples interface (Figure 5-7c). Then, because of the realtime nature of the application, the user can communicate with workers (e.g., "Can you make another one more like your first?") These poses can be imported back into Photoshop to arrange the final result (Figure 5-7d).

As an illustrative example, we simulated a user request to a large number of Puppeteer workers that they pose a human stick figure so that it looked excited. After they finished two puppets, we programmatically prompted workers with a request to

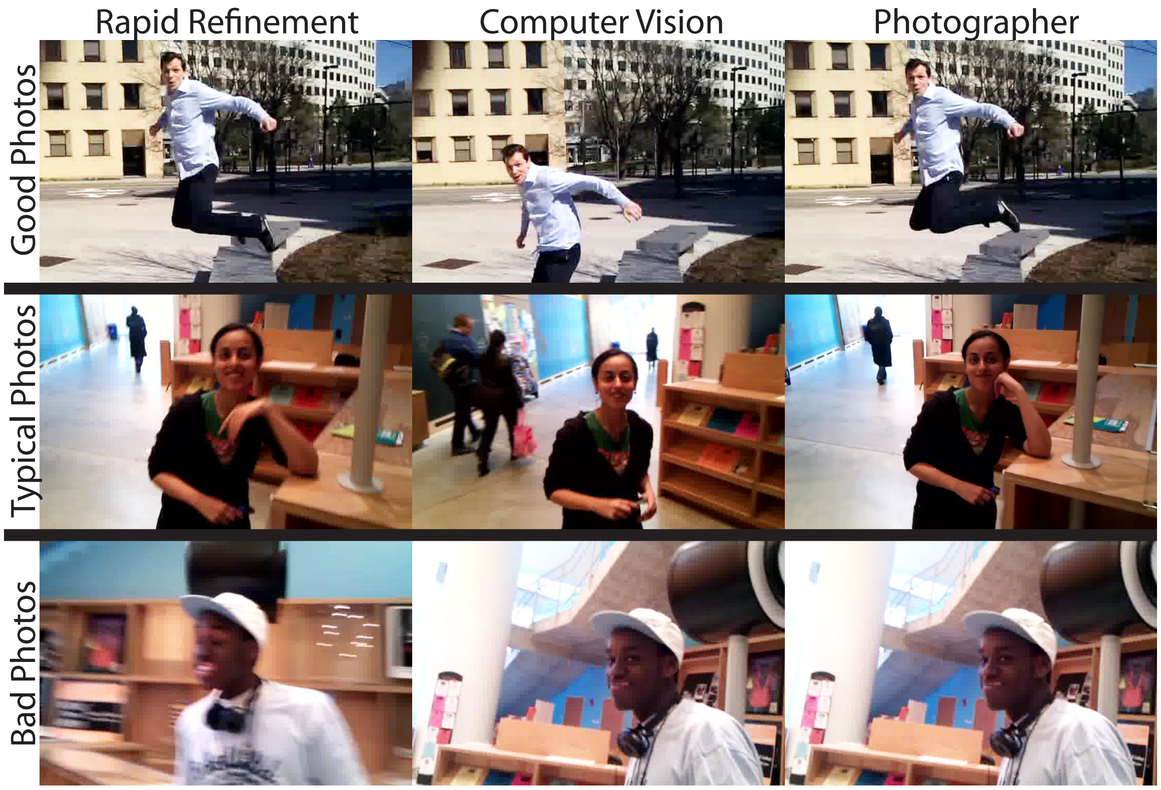


Figure 5-5: Photos from the Adrenaline study. The examples are good, typical, and bad photos that rapid refinement recommended. The computer vision and photographer columns demonstrate how other approaches performed on the same movie.

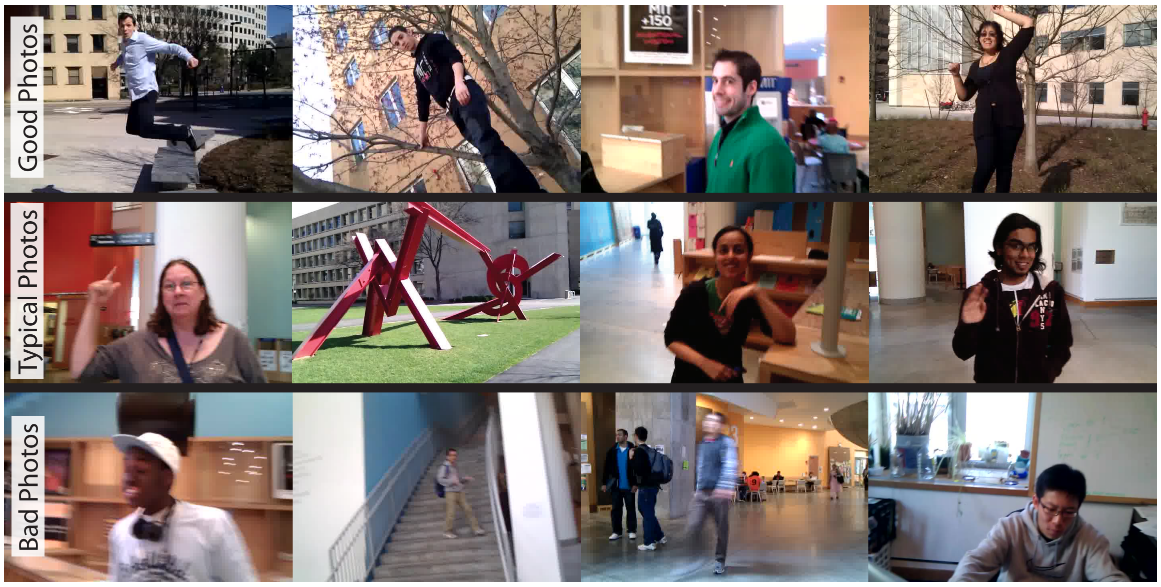


Figure 5-6: More good, typical, and poor photos selected by Rapid Refinement.

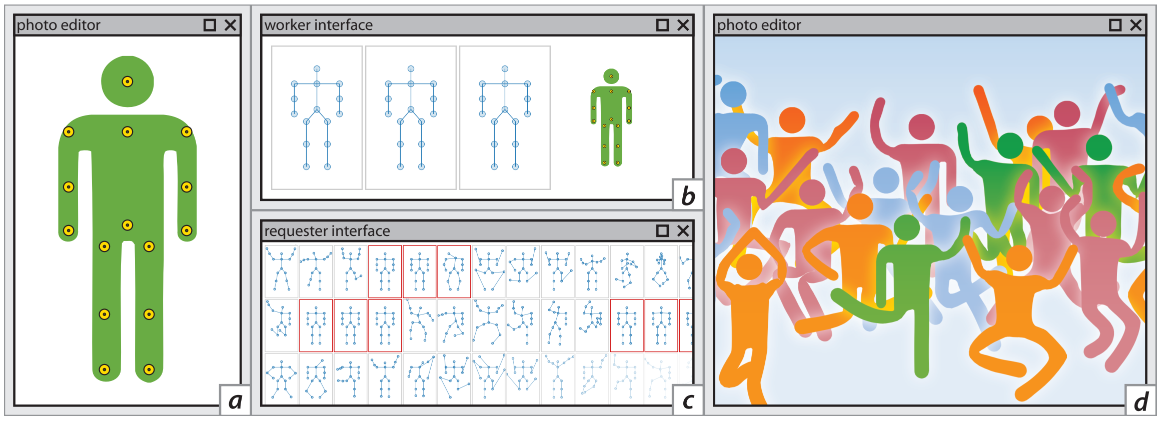


Figure 5-7: Puppeteer allows an artist or designer to generate a group of articulated figures quickly, and to interact with workers as they work. a) The user demonstrates control points. b) Workers move control points to match a request, like "make the person look excited!" c) The user watches incoming results and interacts with workers. d) The final result in Photoshop.

make the third figure look like it is jumping. Anecdotally, the message was quite effective — the majority of workers' third puppets appeared to be jumping whereas the first and second puppets were rarely doing so.

To understand the latency and total throughput of Puppeteer, we repeated the "excited" task, but removed the prompt. Figure 5-7d displays some of the resulting figures. We began the task when there were 8 workers on retainer, received the first control point movement 2.1 seconds later, and received the first completed figure in 25.0 seconds. The first worker completed the entire task (3 puppets) in 46.1 seconds. Workers completed 300 puppets in 16 minutes and 35 seconds, or one puppet every 3.3 seconds. Work output rate was relatively constant throughout.

5.5 Discussion

Having engineered the platform and a set of techniques and applications for realtime crowdsourcing, we can now reflect on open questions and considerations.

5.5.1 Realtime Crowd-Powered Systems

Synchronous crowds and rapid refinement open the door to many applications and techniques. Crowdsourcing has largely been confined to simple, parallel tasks, but synchronous crowds enable coordination and collaboration on a new set of problems. Such crowds can edit documents simultaneously (e.g., [101]), work on team tasks, and distribute large tasks in new ways. Rapid refinement also has applications in other search tasks. For example, rapid refinement might power a predictive mobile web browser that uses crowds to search the page for the user's next action and offer swipe-to-complete for that action, or redesign Soylent's Find task as a synchronous, parallel process.

Larger, non-visual search spaces are another avenue for future work. If it is harder for workers to skim the items or the search space is very large, agreement will be sparser. We are interested in separating tasks into overlapping segments to address this problem. The core interface insight is to stop workers from worrying about deciding on one right answer and instead quickly call out promising areas.

We envision a future where crowdsourcing markets are designed for quick requests, supported by the queueing theory model in Chapter 4. Until we have such a large-scale realtime crowdsourcing platform, scale remains an issue. Our experiments in April 2012 found it relatively straightforward to recruit 30 or more workers on retainer, but if thousands of requesters began using Adrenaline, it might exhaust the worker pool. Given sufficient demand, however, more workers would likely enter the market in exchange for higher wages. Successful realtime services might eventually recruit their own full-time crowds like ChaCha[2].

5.5.2 Rapid Refinement

The Adrenaline evaluation suggests that rapid refinement guides crowds of two to five people to select a good photo faster than even the fastest member of a similar-size group. We might expect that workers would conflict, stalemate and disagree with

[2]`http://www.chacha.com`

each other. However, bottlenecks were uncommon, especially with more than two crowd workers. This result suggests that, rather than interfering with each other, synchronous crowds may hold significant promise for exploring new forms of cooperation and collaboration (e.g., [101]).

Quality may be the most salient issue with rapid refinement: its photos were of reasonable quality, but it did not match Generate-and-Vote. One common source of error was too-fast agreement in the later phases. Sometimes the algorithm decided that workers agreed in the second phase before they had adequate time to explore the timeline and make a decision. We have prototyped designs to compensate for this: for example, requiring that workers explore a minimum fraction of the range before their votes are counted, or requiring a minimum time lapse between phases. These approaches empower the designer to trade off increased lag in exchange for better quality.

A second issue is that rapid refinement uses constants that may depend on task, crowd size and the number of items being explored. For example, constants that work well for small crowds of 3–5 may act differently when 10–15 crowd members arrive. A more principled approach would treat workers' locations as populating a probability density function over frames. Then, measures of distribution peakedness, like kurtosis, would likely ease this problem.

The retainer model was more aggressive than necessary. Often workers joined midway through the process, resulting in more workers than needed. Given a sufficiently busy system, we might re-route latecomers from a retainer into a different task. Alternatively, there may be design patterns that place latecomers into a complementary role in the computational process, like vetting.

5.6 Conclusion

This chapter introduces techniques for realtime crowdsourcing and its applications in user interfaces. Where the fastest crowd-powered interfaces used to be limited by a median response time of nearly a minute [21], we show that it is possible to recruit a crowd within two seconds and complete a complex search in roughly ten seconds. We presented Adrenaline, a realtime crowd-powered mobile phone camera that captures several seconds of video, then uses the crowd to quickly find the best still photo in the set.

Our solution is to introduce on-demand synchronous crowds, where workers arrive and work simultaneously. With synchronous crowds, algorithms can identify regions of likely agreement before workers would normally select a final answer. This intuition led to rapid refinement, a design pattern that focuses the search space on areas of emerging agreement to quickly narrow complex search tasks. The combination of these two ideas enable reliably fast turnarounds for Adrenaline — ten seconds to a preview and the final photo a second or two later. This speed is on average faster than even the fastest individual worker. Finally, we extended the design space of real-time crowd-powered interfaces with Puppeteer, which embedded crowd contributions directly in an authoring interface.

Chapter 6

Beyond Generic Paid Crowds: Specific Data Needs

This thesis so far has focused on 1) tasks that most educated people can complete, such as photo selection and proofreading, and 2) extrinsic motivators such as money. This chapter broadens our scope considerably. Generic crowds cannot complete all tasks a system designer might imagine, for example information needs that need domain expertise. In addition, payment does not necessarily scale easily and it often makes people less interested in a task they would have otherwise found enjoyable [57].

This chapter extends crowdsourcing techniques to gather data that target less common pieces of information and do so without extrinsic monetary incentives. First, in many cases it is possible to design a social computing platform to create a crowd that will produce the data a system needs, especially when generic paid crowds might not have the required knowledge or expertise. We introduce the concept of *friend-sourcing* to encourage this kind of participation. Second, sometimes crowds have already left activity traces across the web, and these traces are sufficient. We will show how large-scale crowd data allows systems to perform *mass personalization* [192] and design interactions for a large number of less common user needs.[1]

[1]This chapter has adapted, updated, and rewritten content from papers at UIST 2009 [14], ACM Transactions on Computer-Human Interaction [15], CHI 2010 [13] and CHI 2012 [20]. Full reports on the evaluations are available in these papers.

This chapter begins by introducing friendsourcing as a technique for gathering information that only members of a small, socially-connected group of individuals will know. Then, it demonstrates how data mining the activity traces from other crowd activities can support a large number of long-tail user needs in interaction. Whereas previous chapters focused on the design of the resulting systems, this chapter focuses mainly on the challenge of motivating participation and mining crowd data. Some details of system evaluations have been compressed to tighten the narrative.

6.1 Friendsourcing

It can be difficult to create crowd-powered systems when only a small network of people is qualified to provide information. In particular, this section investigates the small-network challenge of collecting accurate information about the interests, hobbies, and preferences of people from members of a socially connected group of individuals. This information can be used to personalize users' computing experiences, for example to aid question-answering for topics comprehensible only to a few of a user's friends. Such valuable information is typically held by members of tightly knit groups and its collection poses challenges such as motivating a relatively small pool of people to contribute knowledge. If a crowdsourced system such as Wikipedia only gets 1.6% of its viewers to edit [82], that statistic still results in tens or hundreds of editors for a given page — but when the viewership pool is restricted to the scale of tens or hundreds of individuals as in a social network, a 1.6% hit rate will likely lead to an incomplete and unverified result.

We bring social application design to bear via an approach we call *friendsourcing*. Friendsourcing gathers social information in a social context: it is the use of motivations and incentives over a user's social network to collect information or produce a desired outcome. We shall specifically take on the challenge of friendsourcing for personalization: gathering descriptive information about an individual for use in enhancing computing services for that person. We adapt elements of Games with A Purpose [198] and extend their design principles using social controls.

We shall explore key concepts with friendsourcing applications in the context of two systems: *Collabio* and *FeedMe*.

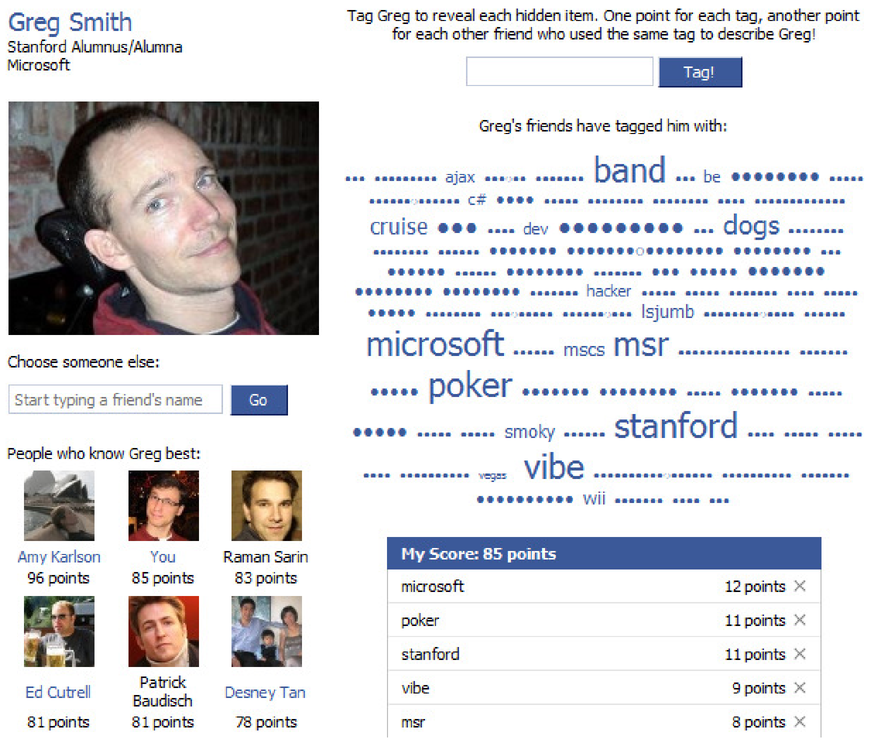
One approach to friendsourcing is to create social interactions that carry information. *Collabio*, short for *Collab*orative *Bio*graphy, is a game we developed to elicit descriptive tags for individuals within the Facebook social network. The game (see Figure 6-1) collects information that friends know about one another, such as peoples' personalities, expertise, artistic and musical tastes, topics of importance, and even quirky habits. The goal is to leverage properties of the social network such as competition and social accountability to solve the tag motivation and accuracy problems within a social framework.

A second approach is to facilitate social interactions that are already happening: *FeedMe* (Figure 6-9) friendsources a personalized news feed by encouraging link sharing between friends and colleagues. FeedMe is a plug-in for the RSS reader Google Reader that supports aggressive content consumers in directed sharing of web content with those who want to receive more but do not want to drink directly from the firehose of the web. FeedMe learns recipients' content preferences based on previously shared content, and suggests potential recipients inline with RSS posts being viewed. Recommendations reduce the amount of effort required to share to two clicks: one click to select a recommended recipient, and one more to send. In parallel, social awareness helps sharers avoid spamming by making visible information such as number of shared items. FeedMe introduces a novel design space within mixed-initiative social systems: the user mediates recommendations not for themselves, but on behalf of someone they know.

6.1.1 Collabio: Social Friend-Tagging

Collabio is a social tagging game embedded in the Facebook social network. It introduces a reciprocal social interaction that generates large term vectors describing individuals on the social network. Collabio users have generated tens of thousands of tags for thousands of individuals on the Facebook social network.

To follow, we describe Collabio's three top level interface tabs: the tab in which



Tag Greg to reveal each hidden item. One point for each tag, another point for each other friend who used the same tag to describe Greg!

Greg's friends have tagged him with:

My Score: 85 points	
microsoft	12 points ×
poker	11 points ×
stanford	11 points ×
vibe	9 points ×
msr	8 points ×

Figure 6-1: The user has guessed several tags for Greg Smith, including *band*, *poker* and *stanford*. Tags guessed by Greg's other friends are hidden by dots until the user guesses them.

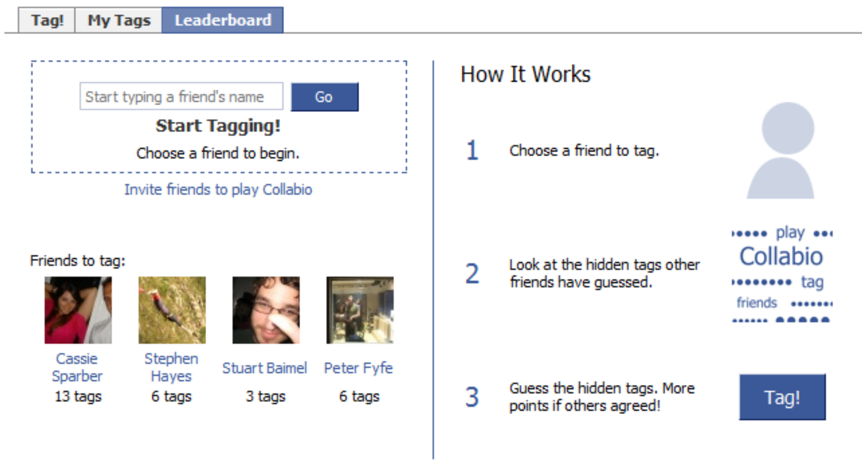

Tag!
My Tags
Leaderboard
Start typing a friend's name
Go
Start Tagging!
Choose a friend to begin.
Invite friends to play Collabio
Friends to tag:
Cassie Sparber
13 tags
Stephen Hayes
6 tags
Stuart Baimel
3 tags
Peter Fyfe
6 tags
How It Works
1 Choose a friend to tag.
2 Look at the hidden tags other friends have guessed.
3 Guess the hidden tags. More points if others agreed!
play
Collabio
tag
friends
Tag!

Figure 6-2: The landing page for Collabio. All application activity occurs on three tabs: *Tag!*, *My Tags*, and *Leaderboard*. The leaderboard is typically below the instructions, shown in Figure 6-5.

users can *Tag!* their friends, the one in which they can manage *My Tags*, and the one in which they can see the *Leaderboard*. We then discuss propagation through the social network, the incentive design space, and issues of cheating and abuse.

Tag Friends

The main activity of Collabio is guessing tags that others have used to describe friends, so the focus of the user's experience is the tagging page (Figure 6-1). The user sees the tag cloud that others have created by tagging the selected friend. When presenting this cloud, Collabio only displays tags that the user has already explicitly guessed (Figure 6-3). Tags not yet guessed are obscured by replacing each constituent letter with a solid circle; for example, the tag *ACM* appears as ●●●. Whitespace in obscured tags is represented by clear circles such as ○. Thus, the length and makeup of the obscured tag provide hints as to the hidden text. As an additional hint, terms in the tag cloud are alphabetically ordered. The tags in the cloud are scaled so that the popular tags are larger.

As the user tags a friend, one of two things happens (Figure 6-3). If the tag is unique and has not previously been placed on their friend, the tag is inserted into the cloud. If the tag exists, then it is revealed within the cloud. For each guess, users

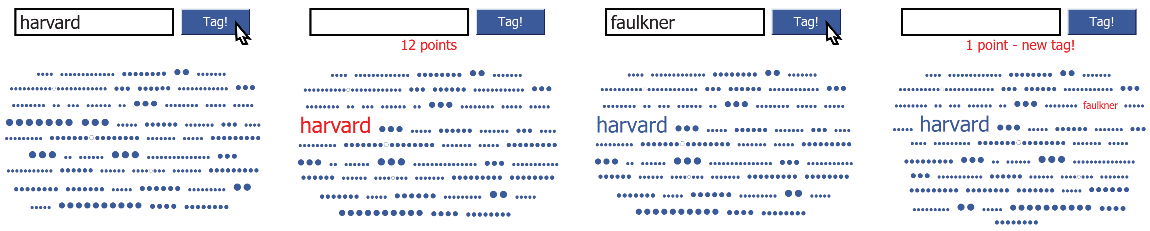


Figure 6-3: The tag cloud begins completely obscured. The player guesses *harvard*, receives 12 points for agreeing with eleven other players and reveals Harvard as a large tag. *Faulkner* is next; it does not match existing tags and is added to the cloud.

receive points equal to the total number of people who have applied a tag, including themselves. If they are the only person to have guessed that tag, then they get 1 point; if there are 11 others, they get 12 points. These points continue to accumulate as more people apply the tag, so earlier taggers' scores rise as well. A user can retract a tag by clicking on a small × by the tag. To expose one's score to others, and to stimulate competition, each tagged friend has a "People who know [this friend] best" pane which lists friends who have earned the largest number of points from tagging that friend (Figure 6-1).

In the current system, if the user is the first to tag a friend, Collabio seeds the tag cloud with terms from the friend's public profile (such as network names, affiliations, or interests), thus ensuring that the tag cloud is never completely empty. These tags are attributed to the "Collabio Bot." We observed early on that users were typically unwilling to tag others who had not already added the application, so this tag seeding is helpful in overcoming reluctance to be the first to tag an individual.

Managing Tags

The My Tags interface allows users to inspect and manage tags their friends have placed on them. The My Tags page contains three sections: a fully uncovered tag cloud (Figure 6-4), an expanded top scorers list, and a table explaining which friends used which tags. In order to allow people to maintain control of tags placed on them, Collabio allows them to easily delete tags from their tag cloud by clicking on a small × by the tag.

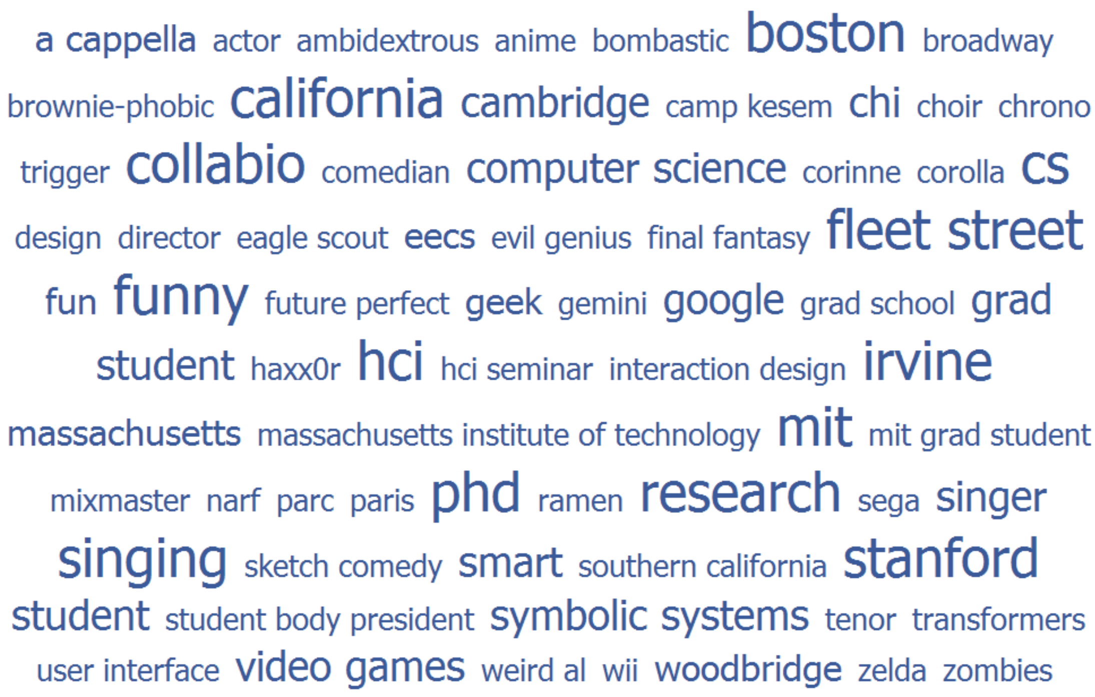

a cappella actor ambidextrous anime bombastic boston broadway brownie-phobic california cambridge camp kesem chi choir chrono trigger collabio comedian computer science corinne corolla cs design director eagle scout eecs evil genius final fantasy fleet street fun funny future perfect geek gemini google grad school grad student haxx0r hci hci seminar interaction design irvine massachusetts massachusetts institute of technology mit mit grad student mixmaster narf parc paris phd ramen research sega singer singing sketch comedy smart southern california stanford student student body president symbolic systems tenor transformers user interface video games weird al wii woodbridge zelda zombies

Figure 6-4: The *My Tags* page allows the user to view their own tag cloud completely uncovered. Not shown: the top 10 scorers list and a complete listing of each tag's authors.

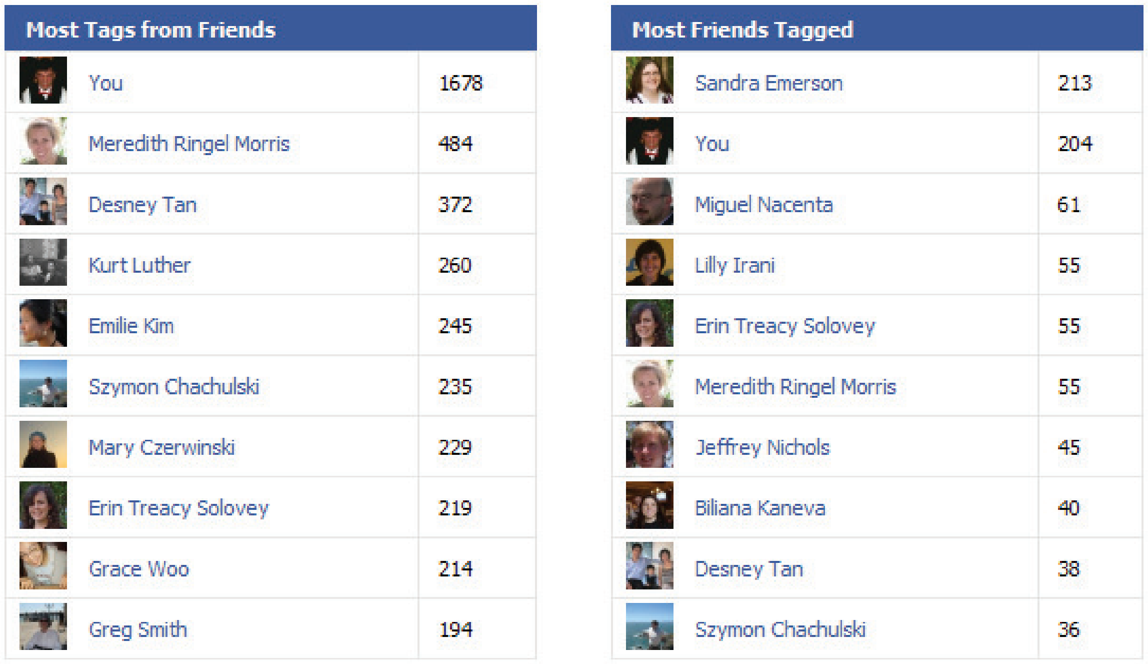
Leaderboard

The third Collabio tab is the Leaderboard. While the individual leaderboards on the *Tag!* tab encourage users to keep tagging a friend until they are listed as one of the Top Friends for that person, the global leaderboards encourage users to continue tagging activity within the application. We present two lists here, one of the friends that have the most unique tags placed on them, and the other of the individuals in the user's social network who have tagged the most other friends (Figure 6-5).

Designing for Viral Spread

Collabio relies on social mechanisms to spread to new users and retain existing ones. For example, the individual leaderboards are labeled "friends who know [this friend] best" to conflate closeness of friendship with score in the game, and notifications purposely do not share all the new tags to entice the user to visit the application.

As with typical Facebook applications, users can explicitly invite others to play.



Most Tags from Friends	
You	1678
Meredith Ringel Morris	484
Desney Tan	372
Kurt Luther	260
Emilie Kim	245
Szymon Chachulski	235
Mary Czerwinski	229
Erin Treacy Solovey	219
Grace Woo	214
Greg Smith	194

Most Friends Tagged	
Sandra Emerson	213
You	204
Miguel Nacenta	61
Lilly Irani	55
Erin Treacy Solovey	55
Meredith Ringel Morris	55
Jeffrey Nichols	45
Biliana Kaneva	40
Desney Tan	38
Szymon Chachulski	36

Figure 6-5: Collabio leaderboards feature the friends with the most tags (left) and the friends who have tagged the most others (right).

More subtly, when a user tags a friend, the application sends a Facebook notification to the friend, whether or not that friend has previously played Collabio. The notification includes the user's name, the number of new tags, and a glimpse of the tags' contents[2]:

> Michael Bernstein has tagged you with **cyclist** and 7 other tags using Collabio. Tag Michael back, or see what you've been tagged with. 2:41pm

A similar version appears on the tagger's wall feed and on Facebook's homepage news feed. Users can also place the occluded version of the tag cloud onto their Facebook profile page. The profile tag cloud demonstrates to visitors the number of tags the individual has acquired and serves as a hook for new users to install and play.

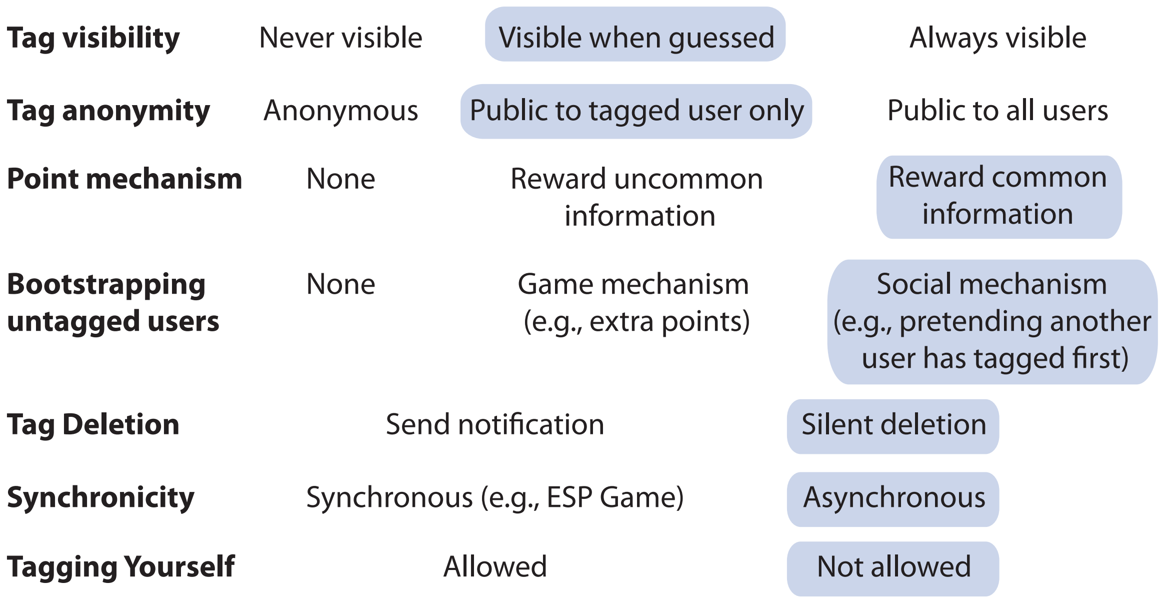
[2]Facebook's changing APIs mean that applications like Collabio would need to choose a separate set of techniques for viral spread today. These techniques were successful in 2008–2010, but several have been deprecated.

Dealing with Cheating and Abuse

Many games suffer from cheating, collusion, or other malicious actions. Because Collabio activity can only occur between people with a mutually-established social connection, we rely on social pressures to prevent this behavior. Specifically, cheating in Collabio would involve annoying your friend by dirtying their tag cloud or sending many notifications, which are undesirable; the tagged individual can also manually retract points or un-friend the tagger.

There are several ways that users could conspire to increase their score. For example, they could ask the person who they are tagging or their friends for the answers. They could also reverse engineer tags using a search strategy on the alphabetized cloud. This behavior does not do active harm to the tag cloud, as it simply reinforces already-existing tags. However, it does erode our premise that popular tags were generated by multiple independent sources. Fortunately, this is more work than just guessing at tags, and it is a poor method for drastically increasing one's score relative to everyone else's since mimicking friends' guesses simultaneously increases their scores as well. Another way to artificially increase one's score might be to tag a friend with a large number of nonsensical tags for 1 point each: e.g., *a*, *aa*, *aaa*, and so on. However, this strategy quickly deteriorates because it does not take advantage of the work others are doing to earn you points and one point becomes worth less and less as more users tag.

Users could also decide to tag an individual with an undesirable tag as a joke or punishment. Since a tag is not automatically revealed to other users until they guess it, the payoff for such a strategy is rather low and non-public, and we did not see much of this in practice. Furthermore, the tagged individual is likely to disapprove of and delete inappropriate tags, thereby eliminating ill-gotten points or reward. We have seen people apply social pressures to friends to discourage such behavior. As regards misspelled or otherwise inaccurate tags, we rely on users' self-interest in maintaining a well-manicured public profile [46].



Tag visibility	Never visible	[✓ Visible when guessed]	Always visible
Tag anonymity	Anonymous	[✓ Public to tagged user only]	Public to all users
Point mechanism	None	Reward uncommon information	[✓ Reward common information]
Bootstrapping untagged users	None	Game mechanism (e.g., extra points)	[✓ Social mechanism (e.g., pretending another user has tagged first)]
Tag Deletion	Send notification		[✓ Silent deletion]
Synchronicity	Synchronous (e.g., ESP Game)		[✓ Asynchronous]
Tagging Yourself	Allowed		[✓ Not allowed]

Figure 6-6: The design space of social tagging applications. Collabio's choices are highlighted in blue.

Incentive Design

One of Collabio's distinguishing characteristics is its incentive system for collecting tags. We designed Collabio's incentive system in a highly iterative manner, controlling and testing dimensions with Wizard of Oz prototypes played as a text-based game over Google Chat. Figure 6-6 summarizes the space of incentive and game options we considered, including tag visibility, anonymity, scoring, and bootstrapping new users. Other applications might choose different tradeoffs in the design space.

Implementation

The Collabio application interface is built as an ASP.NET web application. It communicates with a Microsoft SQL Server-backed Windows Communication Foundation web service for data storage and querying. The application is served as a Facebook Markup Language (FBML) page using the Facebook API.

Field Deployment and Evaluation

We analyzed tag statistics collected between July 2008 and March 2009 (about an 8 month period). In that time, Collabio gathered 29,307 tags (7,780 unique labels) on 3,831 individuals. These tags were generated by 825 different users out of 1,350 who installed the application according to Facebook. The median user who tagged at least one friend received 11 unique tags in return, indicating that even minimal usage of Collabio resulted in a user being relatively well-tagged by friends.

We supplemented this data with a survey methodology aimed at active users, who we defined as users who had tagged at least three friends, were tagged by at least three friends, and had at least nine distinct tags. Using Facebook's notification service, we invited Collabio's 112 most active users to fill out a survey about their experience. Forty-nine users (24 female) responded to the survey. The median age was 27 ($\sigma = 4.1$). The respondents were skewed toward students and researchers with an interest in user interfaces. We offered a small gratuity for responding.

This section reports a summary of the results that are most relevant to crowd-powered systems. Related publications have more details [15].

Tie Strength. Users tended to tag friends close to them: their strong ties [65, 61]. Survey results suggest that users would usually tag closer friends, but not exclusively so. This preference for stronger ties came about because it was easier to invent tags for them and because it could be awkward to send a Facebook notification to a friend who you had not spoken to in months. Our logs show that the users who participated in Collabio tagged 5.8 other friends on average ($\sigma = 13.6$) with 6.1 tags each ($\sigma = 7.3$).

Reciprocity. Social reciprocity through Facebook notifications played a critical role in keeping users engaged. When asked about the reasons for tagging, 82% of survey respondents cited that the friend had tagged them first. In practice, 82% of Collabio users who joined after being tagged reciprocated by tagging at least one of the friends who had tagged them.

Tag Characteristics. In Collabio, single words are often enough to convey the essence of a concept in tag form. Collabio's mean tag length is 8.3 characters ($\sigma =$

Tag Bucket	Definition	Most Popular Information
Popular Tags ($N = 147$)	Three most popular tags for the user	School, workplace or group affiliation (66.0%) Interests or expertise (16.3%)
Middling Tags ($N = 93$)	Less popular than Popular Tags, but occurring more than once	School, workplace or group affiliation (27.2%) Interests or expertise (23.9%) Hobbies (15.2%) Location (10.9%)
Uncommon Tags ($N = 147$)	Occurred only once	Interests or expertise (21.1%) Miscellaneous (15.6%) School, workplace or group affiliation (13.6%) Hobbies (12.9%)

Table 6.1: A breakdown of information type by tag bucket. Affiliation and interest categories were the most popular among the three categories.

5.2). 5,867 tags (75%) are a single word, and 1,913 tags (25%) contain multiple words.

Globally, the tags applied to the most individuals in Collabio are descriptors like *kind* and *smart* as well as affiliations such as *Stanford*. These generically positive descriptors point to the general good-natured bent of most Collabio tags, and suggest that we may have succeeded in preventing large-scale trolling.

To learn more about tag content, we asked each survey respondent to rate nine tags in their tag cloud. These tags were drawn from three buckets (Table 6.1): *Popular Tags*, the three tags applied by the most friends; *Middling Tags*, tags drawn randomly from the set of tags that occurred at least twice but less often than the Popular Tags; and *Unique Tags*, tags drawn randomly from the ones applied by only a single friend. For users who did not have enough tags to fill the Middling Tags category, we instead presented a randomly-generated string and removed the resulting data from later analysis.

For each tag presented, the user provided a rating on a 7-point Likert scale (1 for disagreement and 7 for agreement) for each of two statements: "This is a good tag for me," and "This tag is something I would expect lots of people to know about me." In addition, participants classified each tag into the following categories: school, workplace or group affiliation; professional or academic interest, expertise or title;

	Popular Tags	**Middling Tags**	**Uncommon Tags**
Accurate	$\mu = 6.42$, $\sigma = 0.92$	$\mu = 5.83$, $\sigma = 1.39$	$\mu = 5.13$, $\sigma = 1.61$
Widely Known	$\mu = 6.22$, $\sigma = 1.22$	$\mu = 5.21$, $\sigma = 1.58$	$\mu = 4.14$, $\sigma = 1.77$

Table 6.2: User ratings of how accurate and widely known the tag buckets were, on 7-point Likert scale (1=very inaccurate / not widely known, 7 = very accurate / widely known).

recreational hobby, interest, or expertise; location; personality trait; physical description; name or nickname; another person in the participant's life; inside joke; don't know; or other.

We found that a large percentage of Collabio's tags are affiliations, interests, expertise and hobbies; the long tail of tags contributes a wide variety of unusual information. Table 1 reports that Popular Tags were reported to be mainly affiliations; Middling Tags and Uncommon Tags were more commonly reported to capture interests, expertise and hobbies. The Uncommon Tags were commonly categorized as Miscellaneous, including clothing choices, special abilities, and the name of a friend's dog.

Tag Accuracy and Popularity

Generally, the more popular the tag, the more accurate it was and the more well-known the fact. Survey participants rated all three classes of tags as accurate descriptors of themselves, and all but Uncommon Tags as known by many people (Table 6.2, Figure 6-7). We ran one-way ANOVAs with tag bucket as independent variable and goodness of tag and expectations that others know the given facts as dependent variables. We found significant effects of tag bucket on goodness of tag ($F_{2,384} = 34.5$, $p < 0.001$, $\eta^2 = .15$) and expectation that others know the given facts ($F_{2,384} = 67.1$, $p < 0.001$, $\eta^2 = .26$). Pairwise posthoc comparisons using Bonferroni correction confirmed all factor levels were significantly different from each other in terms of accuracy and anticipated popularity.

We were surprised to find that even the Uncommon Tags were rated as fairly accurate descriptors, with a mean above neutral on the Likert scale. This result sug-

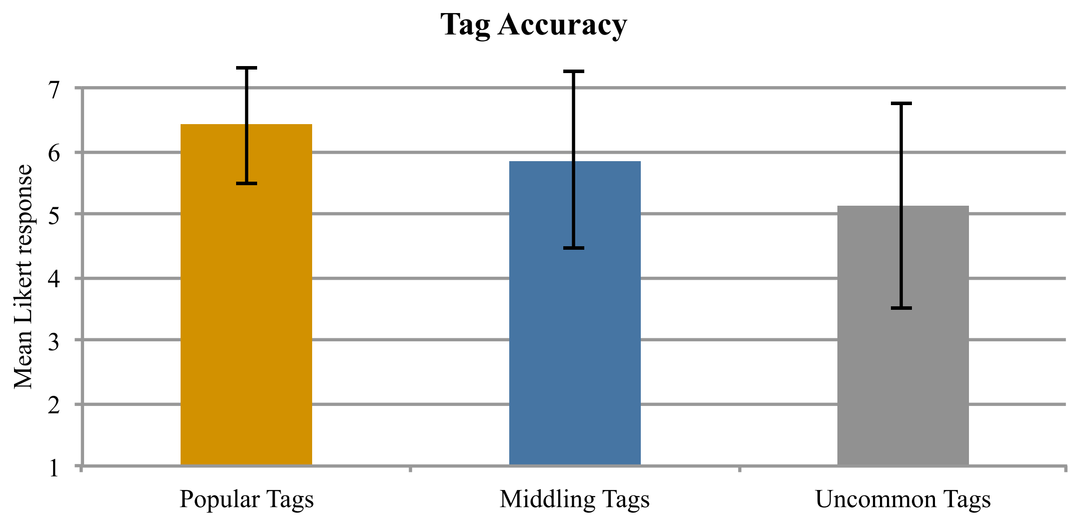


Figure 6-7: A bar chart representation of Table 6.2 indicates that all three classes of tags were rated above neutral (4) on average as accurate descriptors.

gests that there is little inaccurate information in the Collabio tag database. General wisdom in this field would claim that accurate data collection requires repeated independent verification of the same answer, and thus that that one-off answers should generally be discarded [187, 198]. However, we find that even the one-off answers in Collabio (the uncommon tags) are fairly accurate. It seems that Collabio's social incentives help to avoid serious misuse or off-topic tags.

Collabio Tags Are Novel and Unavailable Elsewhere

Our results suggest that Collabio generates accurate tags that are reasonably ordered by importance. However, if these tags are available elsewhere, we have not significantly advanced the state of the art. Could an algorithm or individual outside the social network just as easily create these tags by mining information available in users' Facebook profiles or the web? Could these methods also reproduce the relative ordering of tags?

Rating Study Method. We randomly selected twenty survey respondents from the forty-nine who completed our previous survey. For each survey respondent we utilized the nine tags they had rated in the survey, as well as three *Fake Tags* that were false and thus should not appear anywhere associated with the individual. Fake Tags were chosen from the set of global Collabio tags: one from the top 5% most popular tags, one that occurred less than the 5% most popular tags but more than once, and one that occurred only once. Fake tags excluded any tags applied to the individual.

We recruited four native English speakers comfortable with Facebook and web search, but who had never used Collabio and did not know any Collabio users, to serve as raters. We gave them a brief demonstration of Collabio. The raters' task was to find evidence for each tag on the user's Facebook profile and on the web. For each target individual, raters were presented with the twelve tags in random order and asked to rate each on a 7-point Likert scale according to the following statement: "I can find strong evidence that the tag applies to this individual." Raters were trained to give a score of 7 if the tag appeared verbatim, a score of 1 if there was no evidence in support of the tag, and a score of 4 if moderate inference was required based on the available evidence (e.g., the tag was *Atlanta* but the only relevant evidence was that the person attended Georgia Tech); the other values on the ordinal scale captured in-betweens. Raters were trained on example tags and profile sets until satisfactory agreement on the scoring scale was achieved. We randomized the order that raters viewed individuals.

We tested whether our human raters, as a reasonable upper bound on machine inference, could find the tags on the Collabio users' profiles. Raters rated the set of tags under two scenarios: first using only the individual's Facebook profile available to friends, and second using only web search. In the web search scenario, raters were disallowed from concatenating the individual's name and the tag name into a search query (e.g., "john smith atlanta"), in order to better simulate a tag generation task with no prior knowledge of the tag. We believe this is a more difficult test for Collabio to pass than that undertaken by Farrell et al. [53], who performed string equality tests to see whether tags existed on profiles, because human raters perform semantic inferences.

We also wanted to investigate whether our raters could determine how popular a tag had been, as verified by our survey data. For each individual, we asked raters

	Popular Tags	**Middling Tags**	**Uncommon Tags**	**Fake Tags**
Facebook Evidence	$\mu = 5.54$, $\sigma = 2.36$	$\mu = 4.20$, $\sigma = 2.68$	$\mu = 2.87$, $\sigma = 2.56$	$\mu = 1.56$, $\sigma = 1.76$
Web Search Evidence	$\mu = 5.72$, $\sigma = 2.29$	$\mu = 4.17$, $\sigma = 2.81$	$\mu = 3.04$, $\sigma = 2.65$	$\mu = 1.50$, $\sigma = 1.40$

Table 6.3: Mean ratings applied to tags, from 1 (no evidence to support tag) to 7 (tag appeared verbatim).

to place each tag into its original bucket: Popular Tags, Middling Tags, Unpopular Tags, and Fake Tags. They were told that three tags came from each bucket.

Rating Study Results. Raters evaluated tag evidence on Facebook and the web for a total of 480 tags across the twenty individuals. Cronbach's alpha was calculated to measure agreement across the raters, producing an overall agreement score of .82.

Experts found more supporting evidence for the more popular tag buckets, both on Facebook and the web (Table 6.3, Figure 6-8). A two-factor ANOVA comparing the effect of tag bucket (Popular vs. Middling vs. Uncommon vs. Fake) and evidence type (Facebook vs. Web) on rating found a main effect of tag bucket ($F_{3,1915} = 270.0$, $p < 0.001$, $\eta^2 = .30$), and pairwise Bonferroni posthoc comparisons (all significant $p < 0.001$) suggested that the more popular a tag was, the higher rating it received and so the easier it was to find evidence for. Thus, the more popular the tag was, the more likely it occurred in a publicly visible area. We found no main effect of Evidence type, and inspection suggests that the scores between Facebook and the web are nearly identical.

In the bucket identification task, raters were the most reliable at identifying the extreme buckets: Popular Tags and Fake Tags (Table IV). Raters had the poorest performance on Middling Tags and Uncommon Tags, correctly recognizing only about 40% of each. Thus, beyond the most common tags, it is difficult for non-friends to reconstruct tag rankings.

Overall, raters found evidence supporting Popular Tags, but moderate inference was required for Middling Tags and very little evidence was available for Uncommon Tags. Our original survey respondents indicated that even Uncommon Tags were gen-

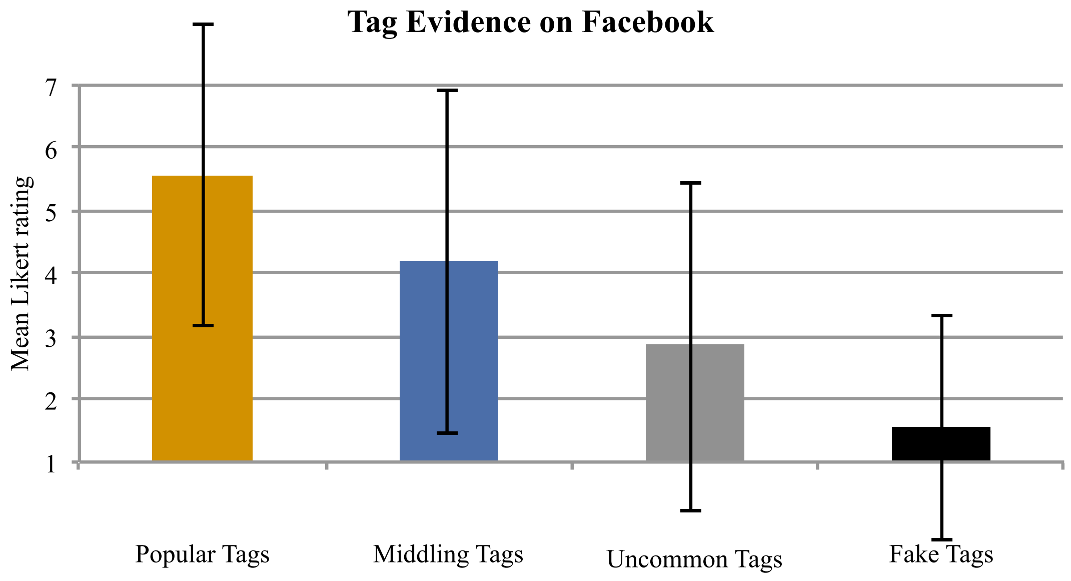


Figure 6-8: A bar chart representation of Table 6.3, focusing on the Facebook condition. Popular Tags tended to have evidence available on the profile; Middling Tags and Uncommon Tags were much less likely to. There was considerable variance in ratings.

		True Buckets			
		Popular	Middling	Uncommon	Fake
Rater Prediction	Popular	**151**	61	24	7
	Middling	63	**94**	50	30
	Uncommon	15	51	**103**	73
	Fake	11	34	63	**130**

Table 6.4: Confusion matrix of rater bucketing decisions. Raters were accurate at identifying Popular Tags and Fake Tags, but less so at Middling Tags and Uncommon Tags.

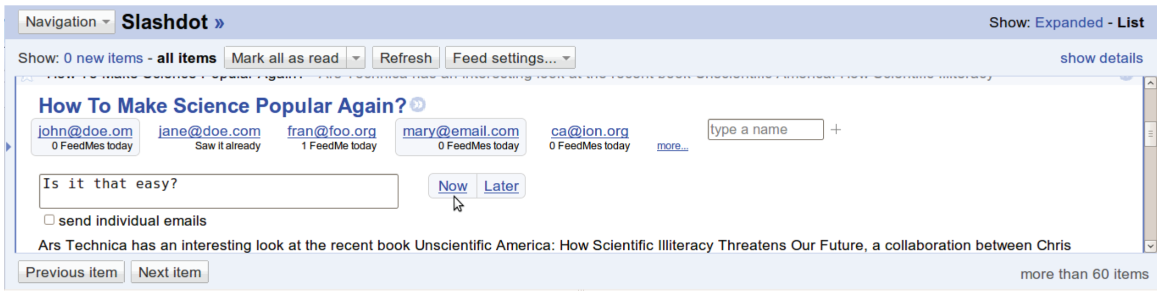
erally accurate, so we may conclude that Collabio is collecting accurate information with Middling and Uncommon Tags that would otherwise be difficult or impossible to acquire. Of the three categories, Popular Tags are fewest in number in the Collabio tag database, so most of the information Collabio collects is unique and thus complements existing public sources with typical online scraping techniques. Raters had considerable difficulty distinguishing Middling from Uncommon tags, and Uncommon from Fake Tags, so beyond the most obvious information it may also be difficult for a human, and certainly a machine, to recreate Collabio's tag ordering even coarsely.

Study Discussion

Tying together the survey and the rating exercise we see that Popular Tags, which largely captured group affiliations, could in principle be generated by mining available information such as Facebook or the web, even though we know of no current system that can do this reliably. Middling Tags and Uncommon Tags, which users view as good descriptors of themselves, are difficult for others outside the social network to verify and by extension to generate. Thus, Collabio generates tags that are not available to typical web mining methods and these tags cannot reliably be judged accurate by individuals outside the social network.

Even unverified, unpopular information is typically accurate in Collabio. This result suggests that guaranteeing accuracy may not be a major design concern for friendsourced systems. This benefit may have been carried over from crowdsourcing: only 1-2% of Wikipedia edits are dedicated to reverting vandalism [105].

Friendsourced applications may be most useful, then, in producing a wide variety of non-generic information about its users. While the system may reward its users for producing popular information, the large majority of tags in our database are not popular. This large majority is the class of information that exhibits the most potential: it is both accurate and unavailable elsewhere. The Dogear Game makes clever use of this situation as well by focusing on incorrect answers as sources of information about the misattributed individuals [49].



How To Make Science Popular Again?

john@doe.om
0 FeedMes today
jane@doe.com
Saw it already
fran@foo.org
1 FeedMe today
mary@email.com
0 FeedMes today
ca@ion.org
0 FeedMes today

Is it that easy?

send individual emails

Ars Technica has an interesting look at the recent book Unscientific America: How Scientific Illiteracy Threatens Our Future, a collaboration between Chris

Figure 6-9: The FeedMe plug-in for Google Reader suggests friends, family, and colleagues who might be interested in seeing the post that you are reading. This user has selected john@doe.com and mary@email.com out of the list of 5 recommendations. The "Now" button sends an e-mail immediately; the "Later" button queues the item in a digest of multiple messages.

6.1.2 FeedMe: A Friendsourced Recommender System

Collabio gathers friendsourced information by creating a new kind of social interaction. By contrast, FeedMe aims to facilitate and ride on an existing social interaction for friendsourced data.

FeedMe takes advantage of the non-uniform (often lognormal) participation distribution on social systems. It hypothesizes that we might be able to incentivize a small number of very active users to participate *on behalf of* the less active users, to everyone's benefit. In other work, we have shown that the minority of active content consumers on the web is also the most interested in routing that information to particular contacts in their network [13].

FeedMe is a plug-in for Google Reader that suggests contacts who might be interested in seeing the content currently being viewed (Figure 6-9), and provides social awareness and feedback mechanisms to ease spamming concerns. To follow, we describe FeedMe's two major components: sharing recommendations, and social awareness and feedback.

Recommendation Interface

FeedMe injects a recommendation interface under the title of every post viewed in Google Reader (Figure 6-9). The recommendation interface suggests individuals with possible interest in the post being viewed. The recommendations make sharing a two-click process: click to confirm the recipient, then click the “Now” button to send an e-mail. Users can optionally add a comment that will be prepended to the e-mail. If multiple receivers are selected, the e-mail goes to all of them; the user also has the option to send separate e-mails rather than cc’ing each recipient.

If interested, the user can display more recommended recipients by clicking “more” to reveal another row of recommendations. If the desired contact has not been recommended or if the user has not shared with the contact before, the user can enter the contact’s e-mail address in an autocompleting textbox. This box is populated with the user’s Google contacts. When the user first uses FeedMe, no recommendations are available and the user must bootstrap using autocomplete. As the user shares, the system recommends past recipients for new posts.

The recipients do not need to be FeedMe users, use an RSS reader, or invest effort in profile authoring. This imbalance is desirable because the majority of recipients do not use an RSS reader, and thus would never use FeedMe. However, many sharers live within an RSS ecosystem. So to train our recommender, we utilize the efforts of the (relatively fewer) FeedMe users: FeedMe models a recipient’s interest by tracking the posts shared with that recipient.

Social Awareness Information and Social Feedback

FeedMe’s social features are intended to display useful information about the receiver to the sharer, give the sharer more control over how the link is sent, and give the receiver a lightweight feedback mechanism.

Load Indicators. To help the user gauge the likelihood of being perceived as spammy, FeedMe provides social awareness information with its recommendations (Figure 6-10). A primary concern is whether the recipient has seen the item already, so FeedMe displays “Seen it already” if the recipient has received the link from another FeedMe user or if the recipient is a FeedMe user and viewed the item in Google Reader. This alert depends on the information that FeedMe can observe, such as FeedMe shares and Google Reader viewership. The interface also helps the sharer gauge how overwhelmed the recipient is by counting FeedMe e-mails from FeedMe

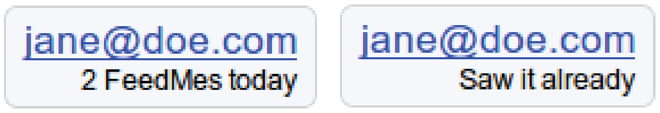


Figure 6-10: Load indicators reflect the number of items sent today (left) and whether the receiver has seen the post already (right).

since midnight. For example, if the recipient has received 2 FeedMe e-mails from one user and 3 from another, the interface displays “5 FeedMes today.”

Digest E-mails. If sharers are worried about sending too many e-mails, they can opt to click “Later” instead of “Now” when sending the e-mail (Figure 6-9). “Later” queues the message into a digest e-mail that is sent out to recipients twice a week when there are pending shared items. A sharer can queue as many items as desired, knowing that only one e-mail will be sent.

One-Click Thanks. Replying to e-mails enables conversation, but recipients may want to express appreciation for the shared post without writing a detailed response. To facilitate this, FeedMe provides a lightweight thanking mechanism to let the sharer know when a recipient appreciates the content. If Dan Olsen were to share a post, a link with the action text “Send Dan Olsen a One-Click Thanks!” is added to the e-mail below the post title. When a recipient clicks the link, he or she is taken to a confirmation page with a thanks leaderboard (Figure 6-11). The leaderboard counts the number of times each of the sharer’s recipients has thanked the sharer, inspired by social games like Collabio. Simultaneously, the sharer is notified of the thanks by e-mail.

Implementation

We implemented the user interface for FeedMe as a Greasemonkey script. Greasemonkey is a plug-in for the Firefox web browser that facilitates the modification of a web site’s code and interface. DOM listeners determine when the user has shifted their attention to a new post. For each post, FeedMe sends an AJAX request for recommendations. The server is implemented using the Django framework and stores data in a MySQL database.

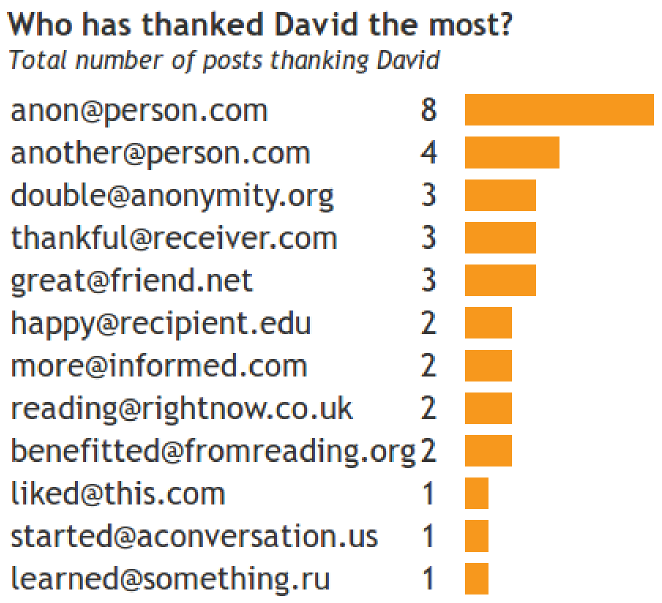


Figure 6-11: The One-Click Thanks leaderboard gives sharers and recipients a chance to see how many other people have enjoyed that sharer's content.

FeedMe constructs a recommendation profile for each user who has received a shared post. To do this, it builds a bag of words model for each recipient composed of words that have appeared in posts previously recommended to them. The algorithm concatenates post title, feed title and content of every post sent to the recipient, then tokenizes the result, performs word stemming, and removes common stop words. Words are weighted by term frequency-inverse document frequency (TF-IDF) [172], so that popular words in posts sent to the recipient are more salient.

The recommendation algorithm uses the standard Rocchio approach, computing cosine distances to each friend of the sharer to the post and ranking the friends' distances [164]. The server creates a TF-IDF word vector for the post, then compares that vector to the vector representing each recipient the sharer has shared with in the past.

Evaluation

To evaluate FeedMe's impact on sharing habits, we performed a two-week field experiment. We recruited 60 participants via blogs and e-mail lists who were regular users

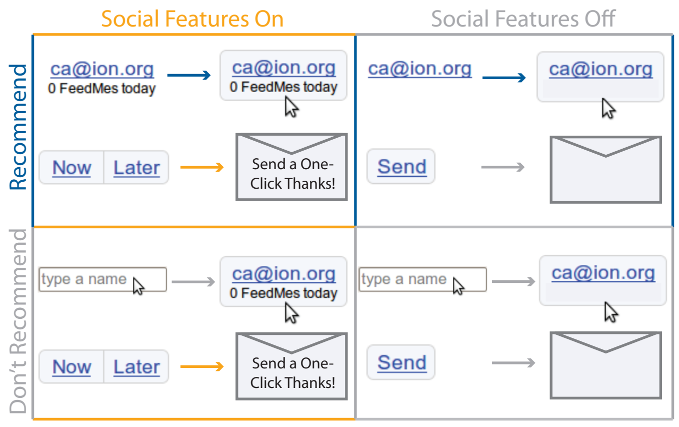


Figure 6-12: FeedMe's evaluation was a 2x2 design. Recommendation features could be turned on or off, and the social awareness features could be turned on or off.

of Google Reader and Firefox. We paid participants $30 for two weeks of Google Reader use with FeedMe installed.

Median participant age range was 26-30, and 46 were male. Many participants were students; others included consultants, designers, an editor, an entrepreneur, a music teacher, a theater technician and a patent agent. Participants also shared 30-day usage statistics that Google Reader makes available before they began using FeedMe. The median participant read 1,598 posts from 52 feeds in the month preceding the study, shared 0 posts from Google Reader using the built-in e-mail interface (though many sent more, max. 224) and publicly shared 5 posts.

Field Experiment Design. FeedMe takes two approaches to facilitate sharing: recommending potential recipients and social awareness and feedback. We designed a study to understand whether these features are useful and how they impact sharing, in a 2 (recommendations) x 2 (social) design (Figure 6-12). All factors were fully balanced and randomized.

Recommendations were either fully enabled or not shown — in either condition, the user could also use an autocomplete textbox to manually add an e-mail address. This factor was within-subjects: participants tried each interface for a week, half receiving recommendations only in the first week, and half receiving them only in the second week. We did not add a second control group with random recommendations: we wished to focus on the social impact of sharing rather than the specific algorithm, and piloting had shown the Rocchio algorithm good enough for eliciting this feedback.

Social features were either fully enabled or fully disabled for the length of the study. Disabling the social features removed information about number of messages received today, whether the recipient had seen or received the link already, the ability to digest e-mails for later, and the ability of recipients to send One-Click Thanks. The social factor was between-subjects, so participants remained in their group for the entire study. We chose to make social features a between-subjects variable to simplify the user experience: four (2x2) configurations would be more difficult for participants to remember and compare.

Halfway through the study and again at the end of the study, we asked participants to complete a survey about their experience. The survey asked Likert scale and free response questions about that week's interface, including ease of sharing and concern about spamminess.

Results. Both sharers and receivers found real benefit in FeedMe. Receivers reported that 80% of shared posts were novel content, and that they were glad to receive the posts. Fully 31% of shared posts had at least one One-Click Thanks. Sharers also enjoyed the tool: 18 participants continued to use the tool a week after the study ended. Participants told us that recommendations made sharing easier and were significantly in favor of it compared to the control interface. Load indicators put sharers at ease and digests freed some users to send many more posts than other study participants.

Of the 60 users who were initially enrolled in the study, 58 used FeedMe until the end of the two weeks and responded to all of our survey questions. These participants shared a total of 713 items using FeedMe, 0.84% of the 84,667 posts viewed while FeedMe was enabled in Google Reader. The median number of viewed posts during the period, normalized out to 30 days, was 1,639 — roughly in line with reading trends prior to the study (median 1,598). Figure 6-13 shows three histograms of usage statistics: unique recipients, shared posts, and recipients per post. There is a right skew to all three distributions: 81% of our users shared with 10 or fewer recipients, most participants shared 20 or fewer posts, and most posts were shared

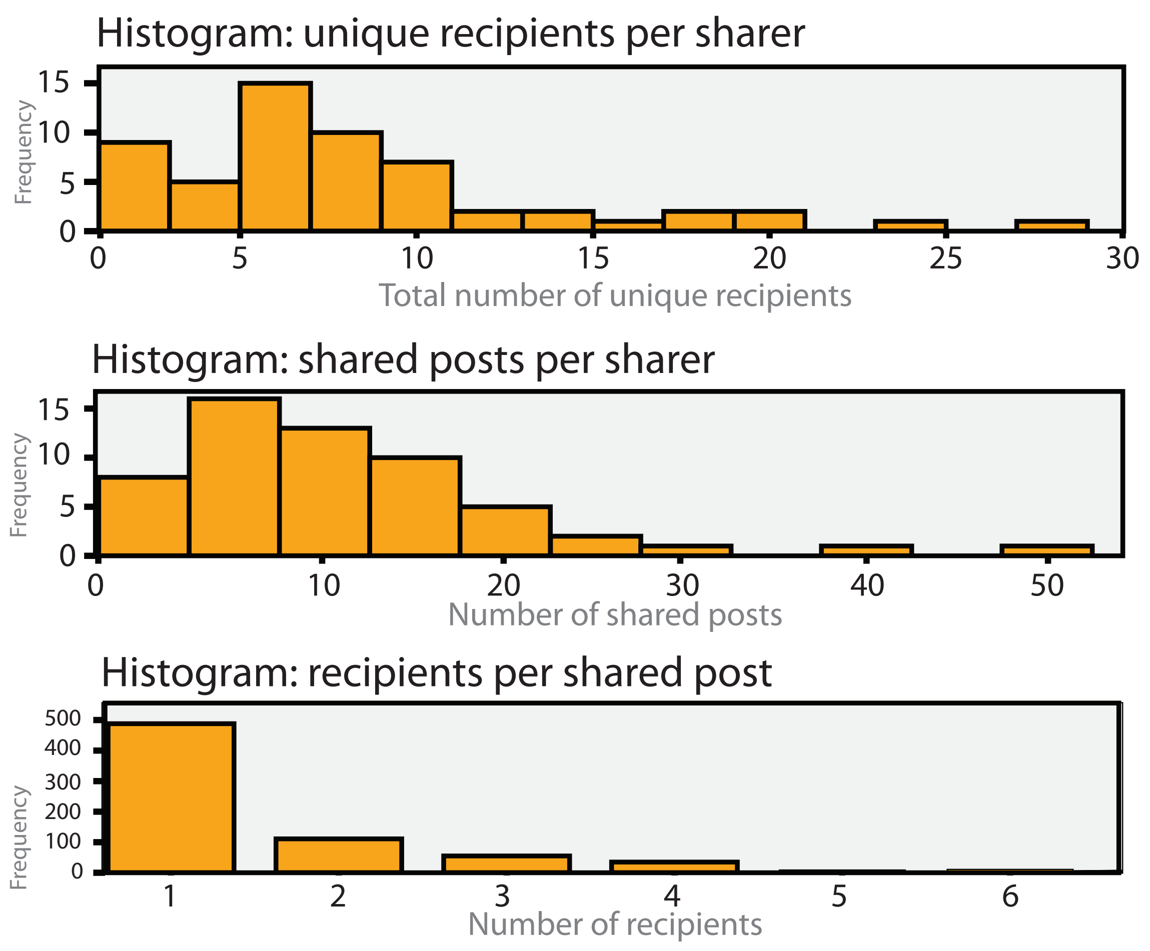


Figure 6-13: Typically, users shared with small numbers of individuals and addressed each message to one recipient.

with a single recipient.

It is tempting to argue that 20 shared posts in two weeks is a low figure, and that participants tried and then discarded FeedMe. Sharers were, however, consistently using the tool. The first two days saw higher activity levels, after which sharers shared a relatively constant number of posts per viewed article through the two weeks (Figure 6-14). We required participants to have the tool installed, but we did not require them to share — the uniformity of sharing across the study suggests that users did not lose interest. As further evidence, two days after the end of the study, 25 of the 60 participants were still using FeedMe to share posts; a week after the end of the study, 18 participants were still using FeedMe. This evidence is indirect, but we consider the voluntary continued usage to be implicit positive feedback.

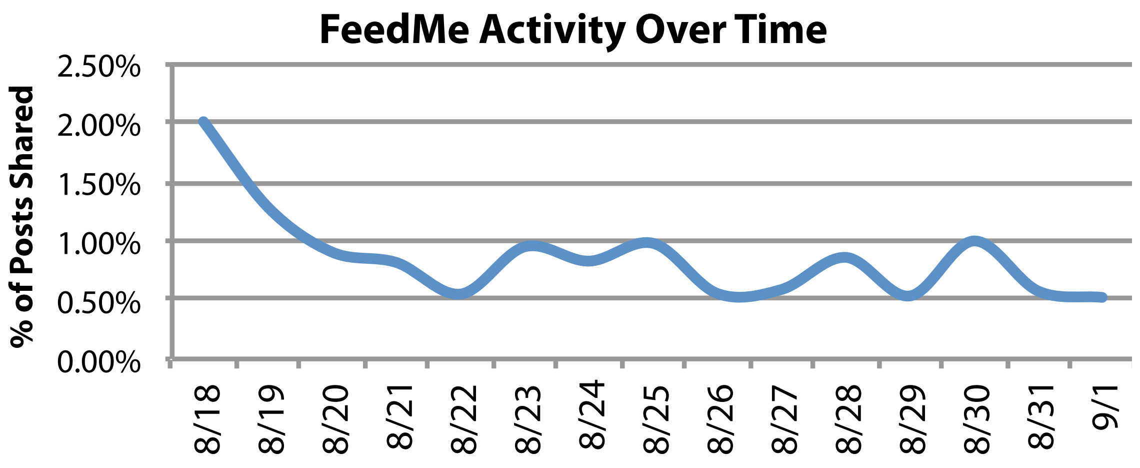


Figure 6-14: After the initial rush of activity, participants continued to use FeedMe to send a consistent percentage of posts viewed.

However, given the relatively small number of shared posts, we proceeded with our summative evaluation largely via qualitative assessments, augmented with usage statistics.

All versions of FeedMe had a large effect on the amount of sharing occurring within the Google Reader interface. A paired t-test comparing the number of posts that sharers e-mailed using Google Reader in the 30 days before the trial ($\mu = 2.7$, $\sigma = .86$) to the number of posts that sharers e-mailed using FeedMe (extrapolated from 14 days to 30; $\mu = 26.5$, $\sigma = 20.9$) is highly significant: $t(57) = 8.447$, $p < .001$. This data is of course not convincing by itself due to the Hawthorne effect, but it suggests that we successfully transitioned information seekers to sharers.

To begin to understand FeedMe's impact, we need to investigate those most impacted by the software. Arguably, this group is not the sharers, but the larger number of receivers who had an unexpected windfall of web links.

Receiver Feedback. Receivers' impressions of FeedMe are an important primary benchmark of success. We emailed everyone who had received at least one FeedMe shared post with a short survey, offering entry in a $30 raffle in compensation. The survey randomly selected up to five posts that the recipient had received via FeedMe. For each post, we asked 1) whether the recipient had seen the link somewhere other than the FeedMe e-mail, and 2) how glad the receiver was to have received that post, on a 7-point Likert scale.

We received responses for 166 shared posts on behalf of 64 receivers. We found that receivers were generally glad to have received the information: the mean Likert response was 5.1 ($\sigma = 1.6$). Receivers also indicated that the vast majority (80.4%) of posts were only encountered through FeedMe. Since the posts were generally enjoyable, it is clear that FeedMe then directly benefited the recipients, who saw more than they would have otherwise.

We conclude that recipients did not feel spammed by FeedMe, were pleased by the shared posts, and were more up-to-date thanks to the novel posts shared by their friends.

Recommendation Interface. Participants viewed the recommendations as a useful means of lowering the effort barrier to sharing. When asked about their favorite part of FeedMe, participants often mentioned the recommendations. One participant appreciated the "keyboard-free, convenient emailing of articles to friends I share with all the time (and have therefore built up a record of in FeedMe)." The recommendations appeared to achieve FeedMe's design goal of accelerated sharing. "I can rapidly click names of people I regularly contact," a participant shared; another reported his favorite feature to be "the speed with which you can share content (without any new tabs or pages)." Participants who preferred the no-recommendation interface did so for reasons of clutter and waste of vertical pixels in Google Reader. FeedMe's recommendations were also occasionally off-target, especially with individuals e-mailed only once.

We asked users to express a preference for either the version of FeedMe that contained recommendations or the one that did not. Using a practice described by Hearst [78], we named the interfaces "Aspen" and "Sierra" for comparison purposes. Two researchers coded the freeform responses as favoring recommendations, favoring no recommendations, or undecided (Figure 6-15). The codings agreed at a .938 level as measured by Cohen's kappa, indicating almost perfect correspondence. A third party arbitrated disagreements. A chi-square test indicates a clear preference for the

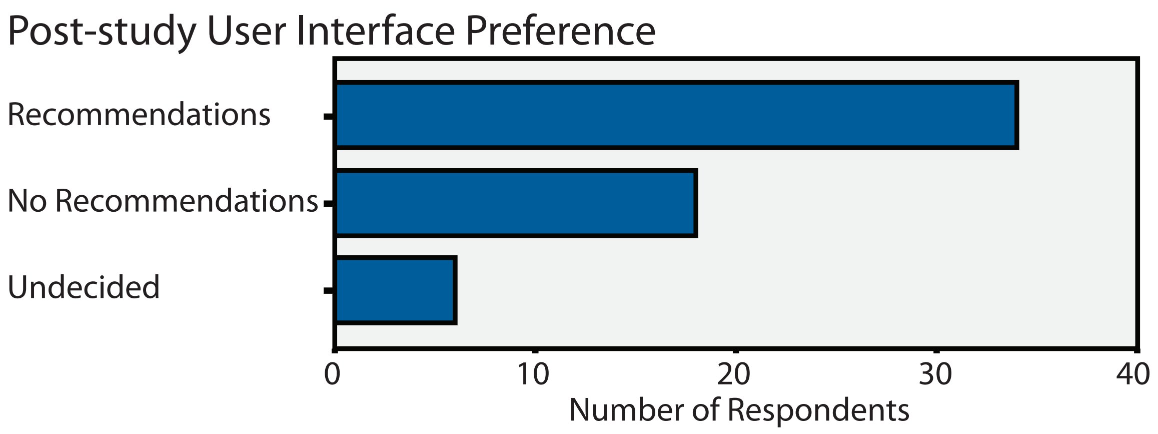


Figure 6-15: Participants reported a significant preference for the recommendation interface ($p < .05$).

recommendation interface ($\chi^2(1, 58) = 4.92$, $p < .05$), with nearly twice as many participants preferring recommendations to no recommendations (34 to 18).

Social Awareness and Feedback. Demand for the social features was high: participants who spent the two weeks without social features (re-)invented them in feedback surveys. Nine of the 30 users with social features mentioned digests, activity statistics, or One-Click Thanks as being their favorite feature in FeedMe. "I could worry less about annoying [my friends]," one participant described. When asked what feature of FeedMe would make them feel more comfortable sharing more, 14 of 28 users without social awareness and feedback indicated that knowledge of how overloaded recipient are would help them feel more comfortable sharing, whereas only 3 of the 30 users with social features made such a claim. The difference between these two groups is significant, as verified by a Chi-Square test with Yates's correction ($\chi^2(1, 58) = 9.34$, $p < .01$). Thus, we believe that the social features went far to address awareness concerns.

Receivers and sharers both appreciated the One-Click Thanks feature. Of 349 shared posts sent in the social-enabled condition, 108 (30.9%) received at least one thanks. An informal sampling of four Facebook feeds revealed that a similar percentage (˜30%) of posts receive at least one Like — an equal engagement from a much larger audience. One recipient who contacted the researchers expressed that One-Click Thanks made it simple to express gratitude for messages which they previously felt pressure to provide an in-depth response to and would typically not respond to at all. The thanks leaderboard did not stimulate competition, but it had the benefit of making user activity visible, thus providing social proof of FeedMe usage.

The Seen It Already indicator was not triggered often because our sharers had largely distinct sets of friends. The feature's usefulness would presumably be improved as entire social circles adopt FeedMe. One participant reported: "I feel like the saw it already' feature could be a sleeper hit for me, it doesn't seem special at first but could be really spectacular to know who has seen or shared an item already." Feedback suggested that it would be particularly useful when sharing with other feed reader users.

Opportunities for Improvement. The clearest concern with FeedMe is related to the choice of e-mail for delivering messages. Some users considered email to be sacred and professional. One shared: "I'm pretty conservative about invading people's email space . . . I worry that they will take real' email from me less seriously" if they also receive lighter, comedic content such as cartoons. The perceived problem is that e-mail is a push medium: recipients are forced to look at the links along with more important information. "Email is a more direct way to communicate," one participant explained, "and I feel that articles that are I read are more like 'ambient' information." For this reason, some power users preferred media they could firehose, such as the public sharing option on Google Reader. Only 5 out of 38 respondents to our original survey indicated that this kind of rate-limiting was their most pressing concern, but it was clearly a theme of the FeedMe feedback. We can think of two explanations: 1) active information seekers are more sensitive to e-mail crowding than average Internet users; 2) FeedMe addressed other concerns successfully enough to make rate-limiting the most pressing remaining concern.

Limitations of the Study

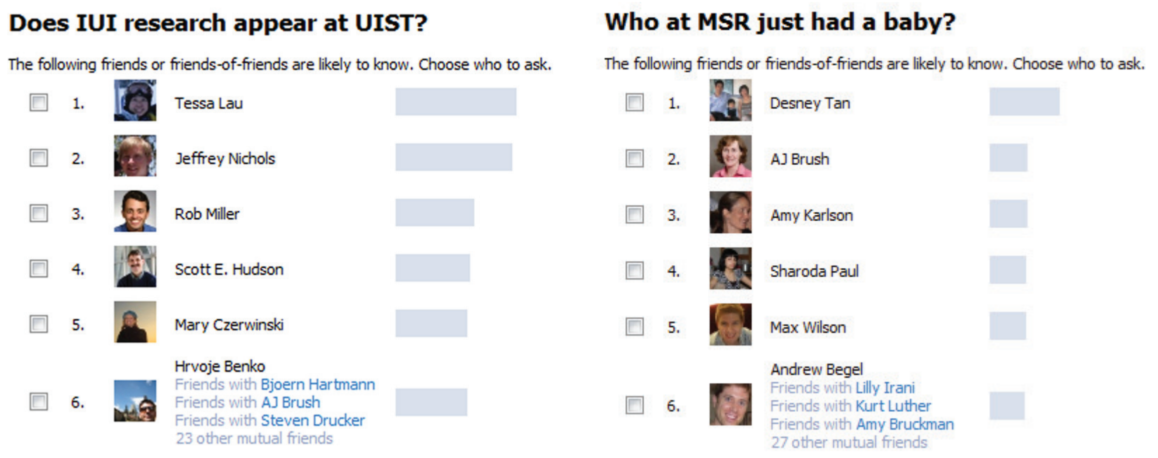
In order to participate in our study, participants had to be Google Reader users with the latest version of Firefox and the ability to install Greasemonkey. Participants who fit this profile are likely to be power users, biasing the kind of users on whom we base our conclusions. Such users often had established norms for sharing with friends, such as mailing lists or IRC channels, and were potentially more sensitive to increasing e-mail traffic to recipients. These biases might have resulted in less sharing in situations where the general population of users might not be so sensitive or have outlets other than email on which to share interesting content.

6.1.3 Systems Powered by Friendsourcing

Friendsourcing is a useful tool for collecting explicit information about preferences and interests that may be difficult to obtain otherwise. Application developers seek out such information when they require user-specific data, for example in recommendation tasks, personalized search, and social network profiles. A strength of this approach is that it can personalized models without the recipients' participation. With FeedMe, for example, we can rely on active RSS readers to install the tool and build the models, because it aids sharing, but the recipients need not do anything in order to benefit.

However, friendsourcing opens another avenue as well: applications which require *social data*; that is, applications which trade in information known to or relevant to only a small group. Yahoo Answers cannot easily help with questions about the history of your small a cappella group or the way your friend delivered his marriage proposal; building applications on data such as the Collabio tags makes this possible.

We have created three illustrative prototypes utilizing the Collabio database: a tag cloud aggregator for tag visualization and exploration, an expert-finding question answering system, and a personalized RSS feed. We attempt two goals with this work: to demonstrate that friendsourced information can reproduce interactions built on asocial sources such as mining of user workstations, and that in some cases friendsourced data can provide new opportunities for interaction.



Does IUI research appear at UIST?

The following friends or friends-of-friends are likely to know. Choose who to ask.

1. Tessa Lau
2. Jeffrey Nichols
3. Rob Miller
4. Scott E. Hudson
5. Mary Czerwinski
6. Hrvoje Benko
   Friends with Bjoern Hartmann
   Friends with AJ Brush
   Friends with Steven Drucker
   23 other mutual friends

Who at MSR just had a baby?

The following friends or friends-of-friends are likely to know. Choose who to ask.

1. Desney Tan
2. AJ Brush
3. Amy Karlson
4. Sharoda Paul
5. Max Wilson
6. Andrew Begel
   Friends with Lilly Irani
   Friends with Kurt Luther
   Friends with Amy Bruckman
   27 other mutual friends

Figure 6-16: Collabio QnA is a question and answer system that uses Collabio tags to find friends and friends-of-friends who can answer your questions.

Question Routing

Friendsourcing gives us knowledge of many more individuals in a social network than we would typically have: the active users have described their inactive friends. Typically, question and answer (QnA) systems such as Yahoo Answers[3] rely on a large community of answerers actively seeking out questions. Expert-finding algorithms can broaden the effectiveness of these tools by actively routing questions to users likely to know the answer. QnA systems with expert-finding components include Answer Garden [1] and Aardvark[4]; Farrell et al [53] suggested that tags could be used for people-ranking.

We embedded this friendsourced QnA system (Figure 6-16) in Facebook. Users ask questions, and the system searches over the collected Collabio tags (or FeedMe term vectors) to identify friends and friends-of-friends who are most likely to be able to answer the question. The user can then choose which friends to send the question to, and Collabio QnA provides a comment board for the answer thread.

Collabio QnA's expert-finding algorithm utilizes the Lucene search engine[5]. Each user's Collabio tag cloud (or FeedMe term vector) is translated into a document in

[3]`http://answers.yahoo.com`
[4]`http://en.wikipedia.org/wiki/Aardvark_(search_engine)`
[5]`http://lucene.apache.org`

the search engine with terms weighted by number of friends who applied the tag. The user's question is then fed as a query to the search engine, and the ranked results are restricted to the user's friends and friends-of-friends. Lucene's default scoring function prefers short documents — in this context, users with fewer tags — so we utilize a length-independent scoring function to give all tag clouds equal scores regardless of size.

Collabio tags and the social network context provide the opportunity for our QnA system to route questions more highly relevant within the user's social network, such as *When is the next HCI group meeting?*, or *Who might be interested in starting an IM football team at Google?* These kinds of questions are difficult to answer using global QnA sites such as Yahoo Answers.

Network Tag Visualization

Collabio has learned thousands of tag clouds for users, so another straightforward step is to create tools to help make sense of the tag space. Collabio Clouds allows users to compare themselves and other users of the system.

Collabio Clouds (Figure 6-17) aggregate tag clouds based on user queries. The user can query his or her own tag cloud as well as the aggregated tag cloud of friends, Collabio users, users tagged with specific Collabio tags (like *tennis* or *Adobe*), or users in Facebook networks or groups. Collabio Clouds allows users to explore questions such as: What do the tag clouds of members of the Penn State network look like? What other tags show up on individuals tagged with *machine learning*? Which tags are most popular amongst all my friends?

Collabio Clouds uses a comparison tag cloud technique developed by ManyEyes [196] to allow users to compare two groups. Thus, a user can compare his or her friends to all Collabio users, compare Michigan students to Michigan State students, compare people tagged with *football* to people tagged with *baseball*, or compare Stanford members of the ACM SIGCHI group to Carnegie Mellon members of the ACM SIGCHI group.

Tag clouds are aggregated by number of members of the group who have a tag, so

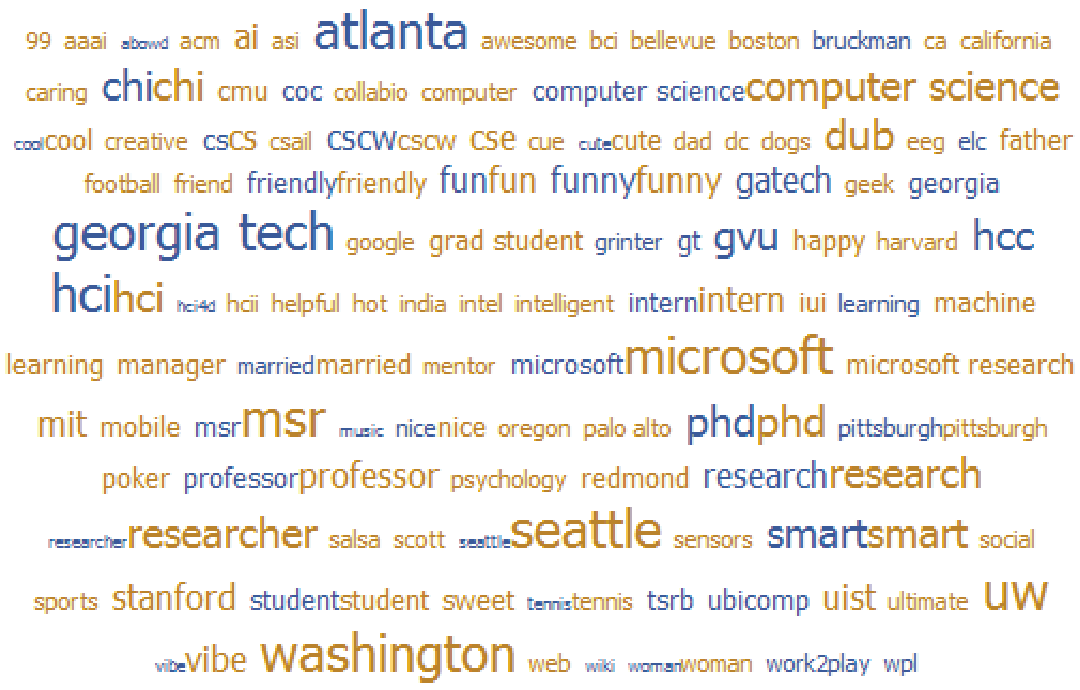


Figure 6-17: A tag cloud comparing users tagged with washington to users tagged with georgia tech in Collabio Clouds.

larger tags are more common in the population. To improve privacy, only tags that are applied to more than one individual are shown in the aggregate tag cloud.

Recommender Systems

FeedMe builds models of its recipients, but Collabio tags allow us to match content to users who have had too few items shared with them. RSS (Really Simple Syndication) is a popular format allowing aggregation of web content, enabling users to subscribe to the feeds of web pages of interest. However, these feeds vary in relevance and can be overwhelming in number, making it difficult to identify the most relevant posts to read.

Collabio RSS is a personalized RSS feed of web content, powered by a user's Collabio tags. Collabio RSS builds on research in personalized web filtering (e.g., [27, 22]). It is unique from most content-based filtering algorithms in that its model is not implicitly learned from user behavior; the tag knowledge base enables a simple information retrieval approach to filtering and enhances scrutability of its results [197].

To produce the news feed, Collabio RSS indexes the title and text content of each feed item as a document in Lucene. When a user requests a personalized feed, it retrieves that user's Collabio tag cloud and performs a document-as-query search on the feed corpus: the weighted tag cloud is concatenated as an OR'ed search query and weighted by tag popularity. Tag weights are log-transformed to prevent the most popular tags from overwhelming the results. We filter the corpus using a sliding time window of the past day and a half to keep the feed's content fresh.

We crawled 2610 popular RSS feeds recommended as bundles by Google Reader, indexing 68,069 items posted over 36 hours. As an example, ten randomly-selected posts vary greatly in topic:

1. 2010 Pontiac Solstice GXP Coupe Test Drive: 28 MPG and Turbo Power, but Practicality Not So Much
2. The X-Files: Season 7: Disc 4
3. 26 Carriers Commit To Deploying LTE; Some Backers Look For Way To Make Voice Calls
4. 5 Reasons Why the PTR Sucks
5. 30 More Free Blog Icons, Website Icons, Symbol Icons
6. 2009 is the year of the comic book in Brussels
7. Superman Cartoons
8. 84th Precinct Crime Blotter
9. 1D-navigation using a scalar Kalman filter
10. 13 Tasteless Costumes Ever

However, Collabio RSS feed identifies items of much greater interest to one of the authors, containing items relevant to HCI, graduate school, and nerd culture in Boston:

1. Weekly Mashable Social Media & Web Event Guide
2. Job Offer: PhD Position in Information Visualization, Vxj University, Sweden, and TU Kaiserslautern, Germany
3. 6 Y Combinator Startups I Would Have Invested In Back Then
4. Mind Meld: Non-Genre Books for Genre Readers *[sci-fi books]*
5. Job: Postdoc in Visual Perception, Delft University
6. The Information School Phenomenon
7. Speaking of (and in) 2009 *[speaking schedule of HCI figure]*
8. Tonight: Video Game Orchestra at Berklee
9. Brain-Computer Interfaces: An international assessment of research and development trends
10. Exploring Siftables: the blocks that play back *[HCI research at author's university]*

The Collabio RSS feed has high relevance because Collabio collects so many tags related to professional and recreational interests. Affiliation-oriented tags, also popular, are responsible for returning news relevant to places and organizations the author has been associated with in the past.

Lessons Learned

This trio of systems powered by friendsourcing have given us insight into techniques and challenges associated with mining friendsourced information like Collabio tags or FeedMe term vectors. Information retrieval techniques such as tf-idf are important means for normalizing out common tags such as *kind*, *beautiful*, and *nice*. Tag sparsity issues may have been expected, but we found that Collabio users typically tried several different versions of a single idea when tagging (e.g., *computer science*, *CS*, *comp sci*), so in practice this was not a major issue. In addition, the stemming that search engines apply to the tags often hashes together different conjugations of a tag.

If sparsity becomes an issue for applications, collaborative filtering (people tagged with one tag were often tagged with another) could implicitly add likely tags.

We cannot distinguish the semantics of any given tag, so we do not know if a tag is appropriate for a given personalization purpose. In the future we intend to try targeting tagging activities more carefully in order to generate tags relevant to a particular application. For example, once a week we might encourage only tags related to college, or to favorite movies. We believe human users are best situated to make these hard semantic decisions, and we would like to leverage this fact. In addition, new tagging tasks might help keep the application fresh.

We believe that Collabio tags will complement existing data mining approaches to personalization. Collabio largely sidesteps the privacy and deployment issues that burden the data mining of private sources such as e-mail or web history. Furthermore, the generated information is guaranteed to be semantically meaningful to the user, whereas automated techniques often result in information that textually distinguishes a user but does not carry much meaning.

6.1.4 Conclusion: Friendsourcing

We have investigated the design space of friendsourced social applications: designs that collect information or execute tasks in a social context by mobilizing a user's friends and colleagues. Friendsouring enables support for previously difficult tasks such as personalization, upkeep of public information about inactive users, and recommendation. To explore this design space, we developed Collabio and FeedMe, two social network applications that extract information about peoples' interests and preferences by encouraging friends to explicitly or implicitly share that information. The resulting data can power visualization and personalization applications, especially those requiring social knowledge.

6.2 Data Mining

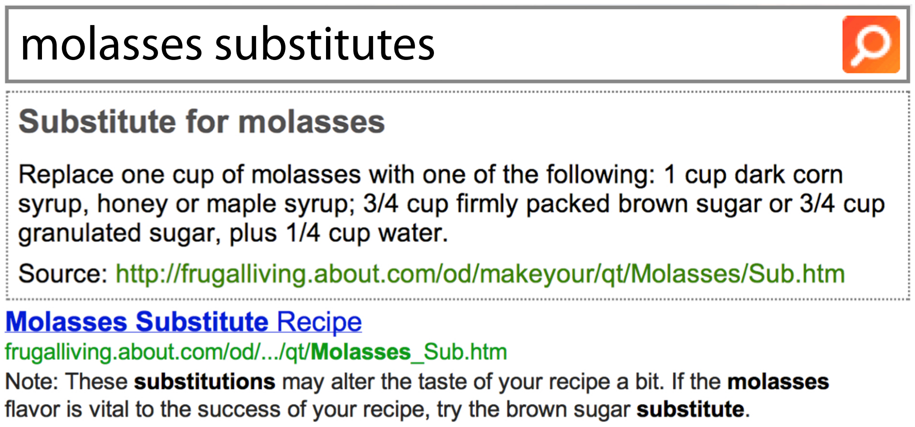
Crowds have already left activity traces across much of the web, for example when they browse or use social media. Rather than designing new crowd interactions, designers might instead use this existing data. This section demonstrates that large-scale crowd data allows systems to perform *mass personalization* [192] and design interactions for a large number of less common user needs. Today, interaction design focuses on identifying a small core set of tasks or goals that the system should support [166]. We suggest that systems can also automatically notice usage patterns in the crowd of users and dynamically adapt to support them.

6.2.1 Tail Answers

Specifically, this section demonstrates how search engines can aggregate user knowledge to improve not just result rankings, but the entire search user experience. We introduce *Tail Answers*, automatically generated search engine results that support a large set of less common information needs. These information needs include the normal body temperature for a dog (Figure 6-18), substitutes for molasses, the currency in Ireland, and many more (Figure 6-19). Each of these needs may occur thousands of times per year, but are too far in the tail of query traffic to be worth assigning programmers, designers, testers, and product management staff to create and maintain answers.

To push answer content down into the long tail (without an exponentially-sized editorial staff), our insight is to aggregate the knowledge of thousands of everyday web users. We turn to web users in each of the three major steps of creating Tail Answers: 1) We identify answer candidates using aggregate search and browsing patterns; 2) We filter those answer candidates to ones which represent directly answerable needs, using search logs and paid crowdsourcing; 3) We extract the answer content from the web, using paid crowds to copy and paste content from the page, then author and edit the final answer text. The entire process can be effectively automated.

Search engine answers and result snippets can have a powerful influence on the


molasses substitutes

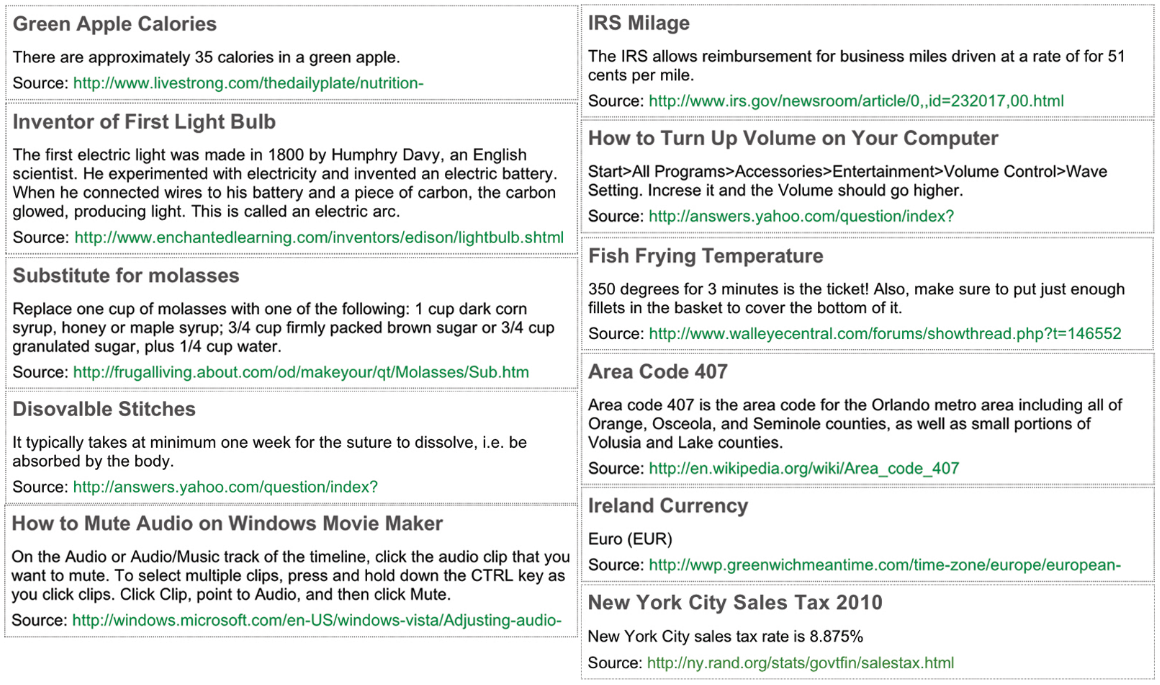
Substitute for molasses

Replace one cup of molasses with one of the following: 1 cup dark corn syrup, honey or maple syrup; 3/4 cup firmly packed brown sugar or 3/4 cup granulated sugar, plus 1/4 cup water.

Source: http://frugalliving.about.com/od/makeyour/qt/Molasses/Sub.htm

Molasses Substitute Recipe
frugalliving.about.com/od/.../qt/**Molasses**_Sub.htm
Note: These **substitutions** may alter the taste of your recipe a bit. If the **molasses** flavor is vital to the success of your recipe, try the brown sugar **substitute**.

Figure 6-18: Tail Answers are inline direct responses for search results. This Tail Answer addresses recipe substitutes for molasses.


Green Apple Calories
There are approximately 35 calories in a green apple.
Source: http://www.livestrong.com/thedailyplate/nutrition-

Inventor of First Light Bulb
The first electric light was made in 1800 by Humphry Davy, an English scientist. He experimented with electricity and invented an electric battery. When he connected wires to his battery and a piece of carbon, the carbon glowed, producing light. This is called an electric arc.
Source: http://www.enchantedlearning.com/inventors/edison/lightbulb.shtml

Substitute for molasses
Replace one cup of molasses with one of the following: 1 cup dark corn syrup, honey or maple syrup; 3/4 cup firmly packed brown sugar or 3/4 cup granulated sugar, plus 1/4 cup water.
Source: http://frugalliving.about.com/od/makeyour/qt/Molasses/Sub.htm

Disovalble Stitches
It typically takes at minimum one week for the suture to dissolve, i.e. be absorbed by the body.
Source: http://answers.yahoo.com/question/index?

How to Mute Audio on Windows Movie Maker
On the Audio or Audio/Music track of the timeline, click the audio clip that you want to mute. To select multiple clips, press and hold down the CTRL key as you click clips. Click Clip, point to Audio, and then click Mute.
Source: http://windows.microsoft.com/en-US/windows-vista/Adjusting-audio-

IRS Milage
The IRS allows reimbursement for business miles driven at a rate of for 51 cents per mile.
Source: http://www.irs.gov/newsroom/article/0,,id=232017,00.html

How to Turn Up Volume on Your Computer
Start>All Programs>Accessories>Entertainment>Volume Control>Wave Setting. Increse it and the Volume should go higher.
Source: http://answers.yahoo.com/question/index?

Fish Frying Temperature
350 degrees for 3 minutes is the ticket! Also, make sure to put just enough fillets in the basket to cover the bottom of it.
Source: http://www.walleyecentral.com/forums/showthread.php?t=146552

Area Code 407
Area code 407 is the area code for the Orlando metro area including all of Orange, Osceola, and Seminole counties, as well as small portions of Volusia and Lake counties.
Source: http://en.wikipedia.org/wiki/Area_code_407

Ireland Currency
Euro (EUR)
Source: http://wwp.greenwichmeantime.com/time-zone/europe/european-

New York City Sales Tax 2010
New York City sales tax rate is 8.875%
Source: http://ny.rand.org/stats/govtfin/salestax.html

Figure 6-19: Tail Answers address less common information needs. These examples (including errors) were produced by the data mining and crowdsourcing processes described in the paper. They trigger on related queries, e.g., *apple calories*.

web search user experience. Nearly half of the abandoned queries in a Google sample displayed a snippet that might have made any additional clicks unnecessary [125]. One quarter of all queries may already be addressed directly in the result page, especially for needs like spell checking, query monitoring, and learning about a term [184]. Successful answers will thus *cannibalize* clicks from the rest of the search results, and searchers will repeat queries to trigger an answer once they learn of it [33]. Even when no answer exists, searchers often use queries for repeated navigation, for example searching for STOC 2012 whenever they want to find the STOC papers deadline [190]. Search result snippets can also sometimes address information needs directly [40]; the snippet for a page, for example, may contain the answer in the text.

Here, we describe how we create Tail Answers to extend search engine answers to tens of thousands of new information needs. Tail Answers are special results inserted inline in the search interface, as shown in Figure 6-18. The Tail Answer contains edited text from a webpage where other searchers found the solution to the same information need. Above the answer text is a concise title to aid skimming, and below it is a link to the source web page for attribution and further exploration. Each Tail Answer is targeted at one particular information need, although it may trigger for many different queries. When a user issues a query that matches a triggering query for a Tail Answer, that answer appears at the top of the search results.

Although answers for popular queries are currently manually programmed, Tail Answers have an automated process to identify information needs that are appropriate for an answer and to author a direct result that addresses the need. In this work, we represent an information need as a set of queries with a similar intent. For example, the queries dog temperature, dog fever, and average temp dog thermometer represent the information need in Figure 6-18. In addition, we assume that Tail Answers can be associated with a web page that contains the answer (e.g., the page `http://www.natural-dog-health-remedies.com/dog-temperature.html`).

To create a Tail Answer, then, our system needs to:

1. **Identify** pages that are answer candidates,

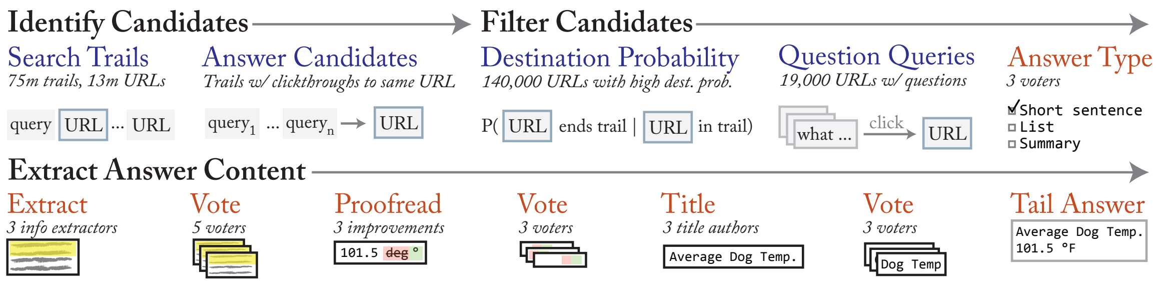


Figure 6-20: An overview of the three phase Tail Answers creation process, which involves 1) identifying answer candidates, 2) filtering the candidates to ones that address "answerable" needs, and 3) extracting the Tail Answer content. Steps that are implemented via data mining are indicated in blue, and those implemented via crowdsourcing are indicated in orange.

2. **Filter** candidates that answers cannot address, and

3. **Extract** the Tail Answer content.

To accomplish these goals, we extract knowledge about answers from the activities of thousands of web users. To identify information needs, we use large-scale log analysis of web browsing patterns. To filter the needs, we augment log analysis with paid crowdsourcing. To extract answer content, we use paid crowdsourcing. Figure 6-20 represents this process visually. We now describe each step in detail, highlighting the technical challenges we solved to improve answer quality.

6.2.2 Identifying Answer Candidates

We begin by identifying information needs, which we call *answer candidates*. An answer candidate is a set of queries associated with a URL from a search result page (Table 6.5). A key idea is to identify browsing patterns that suggest searchers are finding a compact solution to an information need. We use query log analysis to populate our set of answer candidates. To do so, for each search session in our browser logs, we extract a *search trail* [204]: a browsing path beginning with a search query and terminating with a session timeout of thirty minutes. We then group all search trails on the first clicked URL from the result page. For each URL in our dataset, we now have a set of queries that led to the URL and a set of trails that

Page Title	Sample Queries	Type
How to Force Quit on Mac `http://www.ehow.com/how_5178032_force-quit-mac.html`	force quit mac force quit on macs how to force quit mac	Short
Area Code 410 `http://www.areacodehelp.com/where/area_code_410.shtml`	what area code is 410 410 area code area code 410 location	Short
How to bake a potato `http://www.howtobakeapotato.com`	baked ptato how long do you cook potatoes in the oven best way to bake a potato	List
Rummy 500 rules `http://www.rummy.com/rummy500.html`	rules of gin rummy 500 rummy 500 how to play rummy 500	Summary
Pandora Radio `http://www.pandora.com`	radio pandora pandora radio log in	—

Table 6.5: Pages with high destination probability, queries to them, and their crowd-voted answer category. All but the bottom row had a question query: the lack of a question signals that Pandora would not be appropriate for an answer.

describe what users did after clicking through to the URL.

6.2.3 Filtering Answer Candidates

From these answer candidates, we must identify those that are intended for fact-finding [98] and will produce good answers. Some answer candidates have information needs that are too complex to answer; others have underspecified queries where the information need may not be clear. We developed three filters to find promising answer candidates. These filters look for particular types of 1) navigation behavior, 2) query behavior, and 3) information needs.

Filtering by Navigation Behavior: Destination Probability

Our first filter uses the search trails to identify web pages where people quickly end their search sessions. We assume that after a query, people typically end up at web pages containing information that addresses their need. If users stop browsing after they reach a page, that page likely solves the need. If users continue browsing or searching, on the other hand, the page may not succinctly satisfy their need. For example, queries such as *new york times* are often navigational [26]: searchers click on `www.nytimes.com` in the results, then often keep browsing and click on a link to read an article. Other information needs, like buying a new car, are complex and persist across multiple sessions [84], so searchers will typically keep browsing and returning to the search page. But, for web pages like the CHI call for papers, searchers will issue a query (e.g., *chi 2012 deadline*), click through to the page, find what they are looking for, and end their search session.

We formalize the idea of trail-ending web pages with a measurement we call *destination probability*. The destination probability for a web page is the observed probability that a searcher will end their session at that web page after clicking through to the page from the search results. In our search trails, the step immediately after a query is a click on a result web page. If a high percentage of trails end after that click (i.e., if their trail length is two), the destination probability will be high. If most trails instead include actions that return to the result page or browse to other URLs, the destination probability will be low. In other words, the destination probability for a URL is the observed probability that a click to the URL from the search result page is the last action in the search trail.

Web pages with high destination probability are strong candidates for Tail Answers. We filter out any answer candidates that have destination probability of less than 0.3 or fewer than three search trails in our dataset. The 30% cutoff was tuned empirically to balance the number of possible answers (false negatives) with the number of pages with unanswerable content (false positives). Table 6.5 lists five web pages with high destination probabilities. For example, one contains instructions for how to bake a potato.

Filtering by Query Behavior: Question Words

Destination probability identifies pages where searchers appear to be finding immediate answers for their information needs. However, it can be very hard to infer the fact-finding intent from queries that are only two or three words long. For example, an answer for the query *dissolvable stitches* would be valuable if the searcher wanted to learn how long the stitches take to dissolve, but would not if they want to learn the stitches' history.

To avoid this problem, we make use of the minority of searchers who write queries using question words. Question-word queries are useful because they tend to be expressed in natural language, are longer than typical queries, and are more explicit (e.g., *how long do dissolvable stitches last*). These properties make the information need relatively easy to understand. Use of question words also tends to indicate fact-finding intent. We assume that question-word queries often overlap significantly with the unspecified information needs from the other queries, for example that *where is 732 area code* and *732 area code* have similar needs. When this is not the case, we rely on paid crowd members later to disambiguate the most common information need from the set of all queries.

We filter the answer candidates to remove any that had fewer than 1% of their clicks from question queries. The question words we currently look for are: *how*, *what*, *when* and *who*. The bottom row of Table 6.5 demonstrates the kind of error that can occur without a question word filter.

Filtering by Information Need: Answer Type

While question words are useful for identifying answer candidates, neither they nor other types of behavioral log data can help the system understand whether a concise answer could address the information need. Knowing the expected length of the answer is important because crowd workers often extract too much text in order to guarantee that they captured the correct information and thus will be paid. However, overly verbose answers are not useful to searchers. Knowing what kind of answer to expect, for example a short phrase, can help the system perform automatic quality control using length.

To solve these problems, we use paid crowdsourcing via Crowdflower to categorize answer candidates into types. Crowdflower is built on top of Amazon Mechanical Turk and uses hidden quality-control questions known as gold standard questions to filter out poor-quality workers [124]. By prototyping many answers, we developed the following three categories as useful for workers to identify:

- **Short** answers with very little text. For example: "The optimal fish frying temperature is 350F."
- **List** answers, which typically contain a small set of directions. For example: "To change your password over Remote Desktop: 1) Click on Start > Windows Security. 2) Click the Change Password button. [. . .]".
- **Summary or long list** answers, which synthesize large amounts of content. For example, pages requiring deep reading such as "Impact of Budget Cuts on Teachers" and "Centralized vs. Decentralized Organizations".

Workers were asked to read all of the queries that led to the web page, as well as the page itself, and then vote on the best matching category. The third column in Table 6.5 labels each example with its voted answer type.

Although short answers and list answers can be extracted from the web page and edited into an answer, summary answers require more synthesis. For this reason, we leave the generation of summary answers to future work. We use the data about whether an answer is a short answer or a list answer to give workers more specific instructions as they extract answer content and to enforce a maximum number of characters workers can extract from a page.

6.2.4 Extracting the Tail Answer

At this point, we have a set of answer candidates that can be addressed succinctly and factually by the search engine, but each candidate is only represented by a web page and a set of queries. To create an actual answer, we need to extract the information from the web page related to the unifying need, edit it for readability, and write a short answer title. Because automatic extraction algorithms are not yet reliable, we use paid crowdsourcing via Crowdflower.

```
Spooning is a type of cuddling.  When you spoon, you lay on your side
with your back to your partner's chest and the partner behind wraps
his or her arms around you and fits around you like a puzzle.  The
name likely came because of the way two spoons rest on each other,
filling all the nooks.  The "little spoon" is considered the person
in front, the "big spoon" is considered the person in back.  Another
explanation I have read for the origin of the expression:  In days
of old, when a proper young man visited a proper young lady, he was
supposed to do something to keep his hands occupied and away from her
body.  An acceptible activity was sit and carve a wooden spoon while
conversing.  Of a similar vintage, when the couple threw another log
in the fireplace late in the evening, the neighbors would see a burst
of sparks from the chimney, and know that someone was "sparking."
```

Figure 6-21: In this example workers extracted all of the text when an inclusion/exclusion lists was not used. Orange text is the same answer with inclusion/exclusion lists.

The algorithm we developed to guide the crowd to create Tail Answers is as follows. Workers: 1) *extract* (i.e., copy and paste) as little text as possible from the web page using the associated queries as a guide, 2) *proofread* and edit the extracted information into an answer, and 3) *title* the answer descriptively. This information is compiled into a visually distinct search result and presented to searchers who issue the queries associated with the intent, or similar queries. Figure 6-20 contains a graphical representation of these steps.

Worker quality control is a major challenge for the generation of the Tail Answer title and text. Lazy Turkers (Chapter 3) will copy/paste introductory text from each page instead of the answer, and even well-intentioned, pre-qualified workers will extract entire paragraphs or large sections of the page to be sure that it contains the right answer. As a result, early prototype versions of Tail Answers were much too long and of poor quality (Figure 6-21).

One popular quality control technique is to generate a set of potential responses and ask workers to vote on which is the best. For example, we asked three different workers to copy and paste text from the web page and then had five other workers vote to select the best extraction. However, if there are no short extractions, the answer will be long; worse, workers tend to vote for long extractions.

So, it is necessary to add another layer of quality control to help guarantee that the extractions are short and targeted. We adapt the gold standard technique, which requires workers to demonstrate competence by agreeing with the answers to pre-authored example questions for each job [124]. Crowdflower uses gold standard testing by silently inserting gold standard questions into the worker's stream, and only keeps work from people who answer at least 70% of the gold standard questions correctly. Most gold standard tasks involve workers exactly matching the requester's input. For example, for voting we can enforce that workers agree with the authors' selection of which option is the best.

Unfortunately, requiring exact agreement fails for open-ended tasks like extraction. There are often several valid extractions for a page, and it can be just as important to specify which text workers should *not* include. To address this issue, we introduce *inclusion/exclusion lists* for gold standard testing for text generation. To use an inclusion/exclusion list for page extraction, the requester identifies sections of the page that *must* be in the extraction, as well as sections of the page that *must not* be in the extraction, in order for the work to be accepted. By doing so, we are able to tightly scope the areas of the page that are off-limits, as well as information that must be included in the answer for it to be correct. Figure 6-21 is an example of how training workers using inclusion/exclusion gold leads to shorter, more targeted answers.

We implement this technique using negative look-ahead in regular expressions. We also use inclusion/exclusion gold in the title generation step, making sure that workers submit relevant phrases or words and that they do not copy and paste queries verbatim. Inclusion/exclusion gold standards could be useful for other open-ended crowdsourcing tasks like proofreading, replacing expensive approaches such as Find-Fix-Verify as well as qualifier tasks, which cut down on the worker pool significantly.

6.2.5 Implementation

To generate a set of Tail Answers, we began with a one-week sample of browsing behavior from opt-in users of a widely-distributed browser toolbar starting March 22,

2011. We filtered the sample to users in the US who use English when searching. The resulting search trails represent over 2 billion browse events from over 75 million search trails for over 15 million users. We filter pages with too little data by removing ones that have been clicked fewer than three times. Filtering via destination probability and question words resulted in 19,167 answer candidates, including those in the top four rows of Table 6.5.

The query and web page occurrences that make up the answer candidates are distributed similar to power laws, so there are a few pages with many queries and a large number of pages with our minimum of three queries. Answer candidates had a median of three queries ($\mu = 5.2$, $\sigma = 7.4$), 37% of the unique queries contained question words, and the median query had only been issued once in the dataset ($\mu = 7.37$, $\sigma = 35.0$). If each answer candidate were to receive the same number of queries every week for a year as it did during our sample week, the median answer would trigger 364 times per year ($\mu = 1992$, $\sigma = 6318$).

We sampled 350 answer candidates from this set for which to create Tail Answers. We combined several different sampling methods in order to get broad coverage: 100 needs were chosen randomly from the dataset in order to represent the tail more heavily, and 250 were chosen by weighted query popularity to represent query volume.

The number of workers in each stage is a tradeoff between cost and quality. Based on previous experience (e.g., Chapter 3), we recruited three to five workers for extraction and voting. Three workers voted on whether each of the 350 information needs should be addressed by a short answer, a list answer, or a summary answer, for 4.2¢ per need. Of the 350 needs, 146 (42%) were short phrase answers, 127 (36%) were short list answers, and 77 (22%) were summary answers. We focus here just on the short phrase answers, although the process is identical for short list answers and the results are similar. Three workers created extractions for each need (7¢), and five workers voted on the best extraction (10¢). Ten of the 146 answers were voted out by workers for having no good extractions. Of the remainder, three workers proofread the extraction (9¢), and three workers voted on the best alternative (6¢). Three workers authored potential titles (4.2¢), and three workers voted on the best title and filtered the answer if none were appropriate (4.2¢).

At the end of the process, 120 of the 146 short answer candidates became finalized Tail Answers. A number of examples are shown in Figure 2. The cost per answer was 44.6¢ plus a small extra fee for Crowdflower and the expense of the partial results for answers that got voted out. If we were to build Tail Answers for each of the roughly 20,000 candidates in our dataset, it would cost roughly $9,000. This cost can be lowered by combining extraction and title authoring into one task.

6.2.6 Evaluation

In this section, we aim to better understand Tail Answers. Using manual judgments, we show they are high quality and relevant. We then present a controlled user study that shows that Tail Answers significantly improved users' ratings of search result quality and their ability to solve needs without clicking. To remove a source of variation in these evaluations, we focus on the short answers only.

Answer Quality

We first ask whether Tail Answers are high quality. This question has several dimensions: correctness, writing quality, query accuracy, and whether major search engines already have an answer to address the need. We hand-labeled each of the answers with whether the title or the content had writing errors, whether the answer was correct, whether a major search engine already had such an answer, and whether the answer addressed each query in its training set. Two authors labeled each answer; any disagreements were settled by a third rater.

We found that most Tail Answers had high-quality writing in their title and their content (Table 6.6). Of the titles with writing errors, workers had suggested a correct version 50% of the time, but it had been voted down. Likewise, 30% of the contents with an error had a correct version available, but the workers did not vote for it.

Correctness was more variable: some common errors are displayed in Table 6.7. Over two thirds of the Tail Answers were judged fully correct (Table 6.6). A common

	High Quality	**Minor Error**	**Major Error**
Title Writing	83.3%	14.2%	2.5%
Content Writing	82.5%	14.2%	3.3%
Title Writing	68.3%	18.3%	13.3%

Table 6.6: Hand-labeled writing and correctness ratings.

Low-Quality Tail Answer	**Problem**
Resume Writing A Curriculum Vitae, commonly referred to as CV, is a longer (two or more pages), more detailed synopsis. It includes a summary of your educational and academic backgrounds as well as teaching and research experience, publications, presentations, awards, honors, affiliations and other details.	Title does not match the answer
Cary Grant Cary Grant was born on January 18, 1904.	Title does not match the answer
What Reallyahppens.com Most recent WRH radio show from Rense Radio.	Dynamic page has no useful text to extract
Double Irish Tax The Double Irish method is very common at the moment, particularly with companies with intellectual property.	Extracted text is too general

Table 6.7: Examples of common errors in Tail Answers.

minor error (18.3%) occurred when the title did not match the answer: workers who wrote the answer title sometimes paid attention to the original queries rather than the content of the answer. This could be addressed through improved interfaces for the workers and more rigorous quality control in voting. (About 45% of the incorrect answers had a correct version extracted that was not the winner of the popular vote.) Other problems occurred for dead links (i.e., the data could not be extracted) and for dynamic pages (e.g., a "What's My IP?" application and YouTube videos), where workers were unable to signal that the page had no useful information. Two changes would help Tail Answers' accuracy: 1) identifying when dynamic content would make an answer impossible to build, and 2) better quality control to make sure titles are on-topic in the voting stage, since they are written after the answer content.

Fourteen percent of the Tail Answers we generated already had answers available on Bing, a major search engine. Unit conversions (e.g., mL in a tablespoon) were the most common, followed by weather, definitions, and dates. These answers could be filtered in a deployed system, or could be used to replace manually generated answers, which are expensive and time consuming to maintain.

We investigated how closely the answers matched the apparent intent of the queries that represented the intent. (Many queries, like *chi 2012*, may not express the searcher's full intent.) In 58% of the unique queries, it was clear that the Tail Answers addressed the query's intent. About 7% of queries were more general than the answer (e.g., the query was *az municipal court* and the answer gave the phone number to the court), so it is difficult to know whether the answer would have satisfied the information need. Likewise, 23% of queries were generally related to the answer, and the judgment would depend on the exact intent (e.g., a query for *B.C.E.* was associated with an answer for *C.E.*, the Common Era). About 12% of the unique queries were not good matches: about 9% of the queries expressed a more specific need than the answer had (e.g., the query was *fredericksburg VRE* [Virginia Railway Express] but the answer focused on the entire VRE), and about 3% of queries were unrelated to the answer. Often, pages such as the one describing C.E. covered multiple information needs, but workers had to choose just one need for the answer. Clustering these queries into overlapping keyword sets and building separate answers for each would help.

User Evaluation

We also wanted to understand whether Tail Answers positively or negatively impact users' impressions of the search engine result page. In particular, we wanted to know whether Tail Answers improved users' subjective impressions of search results, and whether Tail Answers could compensate for poorer search rankings.

Method. We recruited 361 people (99 female, 262 male) at Microsoft to participate in our study. Most were in their 30s (30%) or 40s (42%), and used search engines hourly (58%) or daily (41%). About 30% held nontechnical jobs. Participants could complete the study from their own computers, and we raffled off $25 gift certificates in return. Participants did not know the purpose of the experiment.

We created a custom version of the Bing search engine that inserted Tail Answers at the top of the search results whenever the user issued a matching query. We gathered a sample of thirty Tail Answers from the 120 we created. Participants were shown five queries, each taken from a randomly chosen Tail Answer, and chose one they found interesting. Participants were required to invent reasons they would issue each query, which is less realistic than showing the Tail Answer when someone has the real information need. However, by giving participants a choice of queries, we hoped they would focus on more personally meaningful tasks. After choosing a query, participants were shown the result page and asked for their level of agreement on a seven point Likert scale with two statements about the search results: 1) "This is a very useful response for the query," and 2) "This page contains everything I need to know to answer the query without clicking on a link."

Our experiment used a two-by-two research design. Each query was randomly assigned either to the *Answer* condition, which displayed a Tail Answer, or to a *No Answer* condition, with no answer. It was also randomly assigned either to the *Good Ranking* condition, where the search engine displayed results ranked 1 through 10, or a *Bad Ranking* condition, which displayed results ranked 101 through 110. In the Bad Ranking condition, the search results were typically much poorer. All conditions appeared to return top-ten results, and we hid ads and other answers. Participants would see each of the conditions randomly as they rated new queries, and were required to rate at least ten queries to be entered in the lottery. At the conclusion of the study, participants filled out a final survey.

We hypothesized that Tail Answers would improve the user experience of the search engine. However, we were also interested in how users would react when Tail Answers fired on inappropriate queries or had incorrect results.

Results. Participants rated 3,963 result pages. Mean ratings are reported in Table 6.8 and Table 6.9. To analyze the results, we used a linear mixed effects model, which is a generalization of ANOVA. We modeled participant, and query (nested in answer), as random effects. Ranking and answer were fixed effects. We also included an interaction term for ranking*answer. This model allowed us to control for variation

	Tail Answer	**No Tail Answer**
Good Ranking	5.81	5.54
Bad Ranking	5.12	3.73

Table 6.8: Mean Likert scale responses to: “This is a very useful response for the query.”

	Tail Answer	**No Tail Answer**
Good Ranking	5.06	4.10
Bad Ranking	4.54	2.66

Table 6.9: Mean Likert scale responses to: “This page contains everything I need to know to answer the query without clicking on a link.”

by answer, query, and user in our analysis. Finally, because participants were more likely to choose certain queries in our dataset, we weighted the observations so that each answer was represented equally in the data. Weighting observations is a common technique when the sample distribution does not match the population; removing the weighting produces very similar results, but we felt that weighting would be the most accurate way to represent all answers equally. We ran the model twice, once for the first Likert scale (1) overall subjective opinion of the result page, and once with the second Likert scale (2) ability to solve the information need without clicking a link.

Tail Answers and result ranking both had significant effects on overall rated result usefulness (Table 6.6). In the statistics to come, we note that weighting the sample leads to non-integer degrees of freedom. Tail Answer appearance, $F(1, 4307.8) = 292.0$, $p < .001$, had an estimated effect of 0.34 points on result usefulness. Good ranking, $F(1, 4306.0) = 570.6$, $p < .001$, had an estimated effect of 0.68 points on result usefulness. Result ranking, which is central to search engines, had an effect size just twice the effect size of Tail Answers: 0.34 vs. 0.68. The interaction was significant, $F(1, 4309.95) = 106.5$, with an estimated effect size of 1.03 points. The large interaction effect indicates that answers are particularly helpful when search results are poor.

Tail Answers were also useful at solving information needs without needing to click through to a result (Table 6.9). The addition of Tail Answers to the search results, $F(1, 4293.0) = 631.4$, $p < 0.001$, had an estimated positive effect of 1.01

points on users' rating. Good ranking, $F(1, 4291.4) = 270.3$, $p < 0.001$, had a smaller effect of 0.50 points on users' ratings, and the interaction term remained large: $F(1, 4295.8) = 60.49$, $p < 0.001$, effect size of 0.91 points. The study design removed other answers from the search results in order to control for variation. It is possible that our effect sizes would be smaller if other answers were included.

Overall, the inclusion of Tail Answers had a positive effect on users' search experience as reflected in their ratings. The impact of Tail Answers was nearly half as much as result ranking, where search engines focus much of their effort. That positive effect was more than doubled when participants were asked whether they needed to click through to a URL. Answers were able to fully compensate for poorer search results, suggesting that a single answer can be as important as good search engine ranking.

Survey Feedback. Participants filled out the survey at the completion of the experiment and provided feedback on the writing, correctness, and usefulness of Tail Answers. Participants found Tail Answers useful ($\mu = 5.8/7$, $\sigma = 1.4$), especially for directed, fact-oriented queries. For many of these queries, Tail Answers addressed the information need directly in the search results. A common theme in the responses was, "it told me exactly the right answer to my question." Participants were enthusiastic that a search engine could answer such unstructured queries. Most participants did not suspect that the Tail Answers were being human-edited.

While participants generally thought the answers were accurate ($\mu = 5.3$, $\sigma = 1.4$) and well-written ($\mu = 5.4$, $\sigma = 1.4$), relevance was a challenge. The crowd tended to create Tail Answers based on the most visible or understandable need in the query logs. When there were multiple information needs on a single URL, the answer would not cover all queries. For example, the only query with clear intent about the Phoenix Municipal Court asked about the court's phone number, so the answer was built around the phone number. However, that answer did not completely address more general queries like phoenix municipal court. In other cases, participants pointed out that the Tail Answer covered the high-level concept but did not have enough detail to fully satisfy their information need. In the future, we believe that it will be important to better target queries either by using the crowd to filter the set of trigger queries, or by A/B testing and measuring click cannibalization [33].

Some participants trusted Tail Answers implicitly, and others wanted more information about sources. Because Tail Answers look like they are endorsed by the search engine, we are particularly sensitive to accuracy and trust.

Generally, participants felt that Tail Answers were concise and well-written. We view this as a success, because extractions in earlier iterations on Tail Answers were much too long. The crowd-authored text had direct readability benefits: one participant remarked that Tail Answers avoided the ellipses and sentence fragments common in search result snippets. Participants occasionally requested richer structure, such as tables and images.

6.2.7 Discussion

We have shown that search engines can cheaply and easily answer many of searchers' fact-finding queries directly. We presented evidence that Tail Answers can improve the user experience, often roughly as significantly as search result quality. Although search engines have used large-scale log data and paid judges to improve search result ranking, our findings suggest that there are new ways human effort can be applied to re-envision the search user experience.

Challenges

Because Tail Answers are presented in a way that appears authoritative, they can potentially spread incorrect or misleading information without oversight. Even simple errors like triggering a Tail Answer on the wrong query can undermine people's trust in the search engine; our evaluation suggested that trimming the query trigger list is an important step for making Tail Answers deployable.

Tail Answers may be particularly tempting targets for search engine spam because of the authority they carry. With Tail Answers, a few members of the crowd would have significant direct control over search results by including advertisements or misinformation. However, a small group of trusted individuals could check for these problems and send answers back if there are problems.

Like result snippets, Tail Answers extract information from web pages and present that content to searchers. Unlike snippets, however, the intent behind the extraction is to fully address the searcher's information need, rather than to direct the searcher to the page. In this way, Tail Answers cannibalize page views. But without the underlying web content, the answers would not exist. To incentivize content providers, one option may be for the search engine to redirect a portion of the query's advertising revenue to pages that provide valuable content. Search engines will continue walking the line between attributing sources and highlighting the most useful information from that source.

6.2.8 Data Mining Extensions: AI, Snippets, and More Answer Types

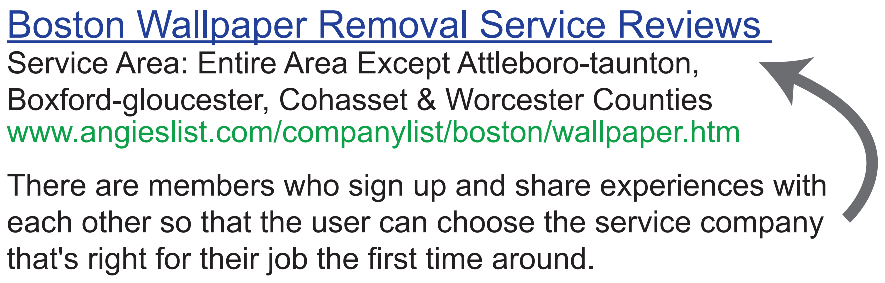
We believe that the insight gained through Tail Answers can deeply extend the vocabulary of search interfaces. We have prototyped several extensions and share some early results in this section.

Artificial Intelligence-Driven Information Extraction

To extract content from web pages and turn that content into an answer, we used paid crowdsourcing. As technologies advance, this balance may shift: automatic systems may assume more or all of the responsibility. Our experiments with automatic systems such as AskMSR [25] and TextRunner [8] suggest that they produce too many poor guesses to be useful. However, a hybrid approach that uses the crowd to vet the answers provided by machine intelligence could be cheap and accurate. To explore this, we connected the AskMSR question-answering system to our dataset of Tail Answer queries, and asked it to generate candidate answers for the question queries. We then used the crowd to vote whether each answer was correct. Table 6.10 demonstrates early results, for example returning "brown sugar" as a substitute for molasses while filtering out highly-rated false positives like "baking". This vote was

Question Query	Algorithmic Result	
	Accepted	Rejected
What is a substitute for molasses?	brown sugar, honey	baking, recipes
What is the cost of mailing letters in the US?	44¢ to 39¢	12, 37¢, mail
Where is area code 559?	State of California	Selma CA, Clovis
How much nicotine is in a light cigarette?	Low density, 6mg	milligrams, 14mg

Table 6.10: An automated question-answering system proposed Tail Answers and crowds filtered them.


Boston Wallpaper Removal Service Reviews
Service Area: Entire Area Except Attleboro-taunton, Boxford-gloucester, Cohasset & Worcester Counties
www.angieslist.com/companylist/boston/wallpaper.htm

There are members who sign up and share experiences with each other so that the user can choose the service company that's right for their job the first time around.

Figure 6-22: The Tail Answers crowd extraction algorithm (bottom) can suggest replacements for result snippets (top).

much cheaper than paying for extraction and proofreading.

Smart Snippets for Popular Queries

In addition to standalone answers, the crowd can help with snippets, the short page summaries that appear underneath the page title in search results. Instead of tail needs, popular queries are a good match for snippet improvement because they are seen by a large number of searchers. In particular, we focus on popular queries that have high click entropy (i.e., people click on many different results for the query). Queries like *wallpaper* have high click entropy because they have multiple meanings (e.g., computer desktop art versus home wall decoration), and searchers may not have enough information scent [160] in the snippets to make good choices. We can use the extraction routine from Tail Answers to find snippets for these sites. Figure 6-22 demonstrates the resulting improvements to a high-visibility search snippet for the query *wallpaper*.

New Classes of Answers

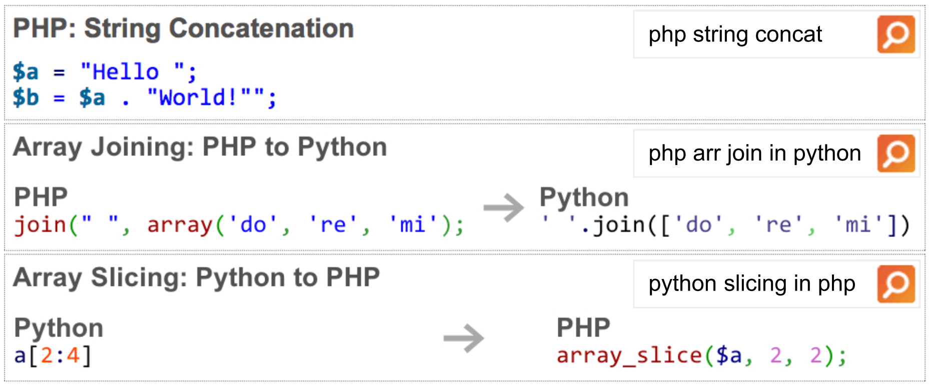
We have thus far explored short and list-style answers, but there are many more possible answer types that could be developed with our approach. For example, answers could be created to help users achieve high-level goals like creating a website or planning a vacation to Yosemite [123, 207]. They could also summarize web content, automatically create answers for spiking queries or news stories, or even connect searchers with other users who might be able to help solve their information need [84]. To create more sophisticated answers, we expect to transition from generic crowd workers in Mechanical Turk to more expert workers like those found on oDesk. We could also give other searchers the ability to edit the answer, much like Wikipedia. The amount of effort and cost could be applied differentially, based on potential gain, with more invested in more popular or high impact information needs.

Because Tail Answers are general-purpose, it is impossible to provide custom user interfaces. However, if we focus on a particular set of information needs, we can build special user interfaces and data extraction requirements. Figure 6-23 shows example answers we have built for translating commands between programming languages, for example understanding how to translate PHPs array join syntax into Python. We began with a list of programming primitives in Python, then asked workers to volunteer the mapping into PHP. With this mapping, the Tail Answers can return results for functions in either language, as well as translate between the languages, with a specially designed interface.

Destination probability can also help identify new kinds of answers. For example, pages with telephone area codes tended to have high destination probability. Armed with this information, search engines might start building answers specifically for area code queries.

6.2.9 Conclusion: Data Mining

By mining past crowd activities rather than creating a new crowd for each system, system designers can develop interactive systems that draw on less controlled but


PHP: String Concatenation

php string concat

```
$a = "Hello ";
$b = $a . "World!"";
```

Array Joining: PHP to Python

php arr join in python

PHP
```
join(" ", array('do', 're', 'mi');
```
Python
```
' '.join(['do', 're', 'mi'])
```

Array Slicing: Python to PHP

python slicing in php

Python
```
a[2:4]
```
PHP
```
array_slice($a, 2, 2);
```

Figure 6-23: Code tutorial answers. Within a domain, Tail Answers like these can specialize their user interface.

more naturalistic behaviors. For example, Tail Answers can provide succinct inline search results for less frequent and extremely varied information needs. To build Tail Answers, we draw on the aggregate knowledge of thousands of web users. We mine large-scale query logs for pages that tend to end search sessions, select candidates where searchers have used information key terms like question words, and use paid crowds to remove candidates that cannot be answered succinctly. Finally, crowds extract the information from the web page, edit it, and title it. Our evaluation of Tail Answers demonstrates that they can significantly improve the search user experience and searchers' ability to find the information they are looking for without navigating to an external web page.

6.3 Conclusion: Beyond Generic Crowds

This chapter demonstrates that crowd-powered systems can look beyond generic, on-demand paid crowds. Specific crowds have specific motivational levers that the designer can engage, such as social interaction with friendsourcing. Data mining can also provide access to naturalistic user behavior in a wide variety of scenarios that designers and developers would not have foreseen. These techniques collectively have broad applications to the design of interactive systems. They allow users to draw on thousands of professional authors' styles to improve their own writing; to galvanize gigabytes of open-source code to auto-complete not just the line of Python they are writing now, but the entire design pattern they are trying to apply; to mine social network status update feeds to personalize search; to accelerate navigation through a popular but poorly-designed web site.

Chapter 7

Discussion: Framework, Limits, Ethics

This thesis lays out a vision and implementation strategies for interactive designs that move beyond the user-system divide by reaching out to crowds. These crowds can be paid, incentivized through social systems, or mined from data. Having laid out this vision, we now have the opportunity to reflect on the future of this research and of the field.

Four topics are particularly relevant: a future of deployable wizard-of-oz prototypes, a design framework for crowd computing and different kinds of crowds, the limitations of this approach, and the creation of an ethical framework to guide crowd computing.

7.1 Deployable Wizard-of-Oz Prototypes

Wizard of Oz prototyping has long been a technique in user interface design and artificial intelligence [99]. In Wizard of Oz experiments, the developer or experimenter simulates parts of the computer program that are not implemented yet. This allows the designer to gather feedback on a system before putting in days or months of development time to build it. For example, Kelley completely simulated a natural language input interface in order to gather training data to author its grammar [99].

However, Wizard of Oz systems were always just prototypes: the wizard would need to be removed from the system for eventual deployment.

Our work suggests that it may be possible to transition from an era where Wizard of Oz techniques were used only as prototyping tools to an era where a "Wizard of Turk" can be permanently wired into a system. This change allows us to deploy useful applications today and use them to change lives for the better. Then, the system can use the crowd's results for training data and gradually transition to automated methods.

Aardvark[1] used this pattern [60]. As a social search start-up, it needed to route questions to individuals who were qualified to answer. So, the employees began by routing questions manually. Manual routing supported a growing user base and produced training data. Then, when the algorithm was sufficiently trained and tested, the developers began using it.

There are cases when a deployable Wizard of Oz system might not work. If many people find the system useful, there might not be enough crowd members to satisfy demand. There are also tasks where crowds might not perform well enough to be usable: arguably, domains like music search from an audio sample might fall into this category. Finally, even realtime crowds might not be as fast as an algorithm eventually would be, so it would be difficult to use crowds to prototype systems that require sub-second answers.

If this effort is successful, there may be many crowd-powered systems deployed in the coming years. These systems will need a design framework — the next section begins to construct this framework.

7.2 Design Framework and Tradeoffs

Crowd-powered systems combine two user interfaces: the interface shown to the crowd, and the interface shown to the end user. The design space of crowd-powered interfaces must consider both user groups.

[1]`http://en.wikipedia.org/wiki/Aardvark_(search_engine)`

Crowd	
Motivation	What incentive does the system use to encourage participation? Is the crowd intrinsically or extrinsically motivated to participate? If intrinsic, are the motivations social, fun, fame, glory?
Quality Control	How does the system automatically detect and filter out poor crowd contributions?
Crowd Size	How many people are in the crowd? How many need to participate?
Participation Distribution	What is the desired distribution of contributions? Can a few participants contribute most of the content, or does participation need to be more evenly distributed?
Temporality	Is the crowd expected to react in realtime?
Collaboration	Can the crowd collaborate on the work, or is the work distributed and individual?
Expertise	Does the task require crowd members to have a specific expertise, like knowledge of a particular subject? Or, can most Internet users complete the tasks?

Table 7.1: Design dimensions for creating the crowd.

7.2.1 Crowd Design Tradeoffs

Table 7.1 describes the decisions that a system designer makes when considering the kind of crowd to use.

Motivation is one of the most important decision points. Labor economics and social psychology divide motivations into *intrinsic* and *extrinsic* classes. Intrinsic motivations are ones which derive from enjoyment in the task or its results, and they typically follow from giving people a sense of autonomy or mastery in the task. Extrinsic motivations are based on outcomes that people participate in order to reach: money, fame, or a grade. Social psychology has demonstrated that extrinsic motivations like money actually *undermine* the intrinsic motivations to participate [57]. This literature suggests that paid workers will (on average) work less hard than workers who are motivated by fun or inherent interest.

As a result, it becomes important to target the motivation to the goal. The simplest situation is if crowds are already participating in the activity that the system requires — web search, in the case of Tail Answers. The next step up in the hierarchy is to create a new kind of activity which crowds would be intrinsically motivated to do. For example, friendsourcing is a new avenue to express a desire to communicate with friends or demonstrate deep knowledge of their interests. NASA ClickWorkers [95] and the Search for Jim Gray [81] allowed participants to work towards a goal of importance. Games with a Purpose [198] and FoldIt [37] translate typically boring tasks into a frame that taps into the latent desire for fun and play. If interest or fun is not enough, then quantifiable extrinsic motivators like points, badges, or fame can keep the top users continuing to participate. Games with a Purpose use several of these motivators, as do question and answer sites like StackOverflow[2] (reputation) and crowd-contributed knowledge bases like Wikipedia (number of edits, barnstars [111]). Only in the case where none of these would be possible, for example editing a college student's paper, should extrinsic motivators like money be used. Once money is introduced into the equation, social psychology dictates that all other motivators matter much less.

An intrinsically motivated community and organic growth can be difficult to achieve. Creating a community of intrinsically motivated users from scratch is difficult, and many communities never reach critical mass. In this case, it may make sense to carefully adapt the motivational structure of a community over time.

Quality control refers to the system's ability to automatically identify the best crowd contributions. Does it use other crowd members to vet, as in Find-Fix-Verify's Verify step? Does it look for independent agreement and risk a regression to the mean, like Rapid Refinement, Iterate-and-Vote [128], or von Ahn's input-agreement games [198]? Is the task framed in such a way that there is an objective answer and workers can be tested against ground truth?

The crowd systems presented in this thesis rely mostly on independent agreement and peer vetting. One strength with this approach is that, anecdotally, crowds are better at evaluating contributions than they are at producing those contributions. For example, in a separate experiment where we asked crowds to rewrite news headlines in the style of Dr. Seuss[3], only a few suggestions were good. However, the crowd

[2]`http://www.stackoverflow.com`
[3]`http://groups.csail.mit.edu/uid/deneme/?p=638`

was able to identify those high-quality suggestions. However, these approaches direct systems toward responses that everyone can agree on, which can push out unusual but creative responses. This is the same reason crowd-voting sites such as Reddit are full of generic content (e.g., LOLcats): almost everyone finds cats cute, and more interesting but niche-directed content cannot gather enough votes. Moving forward, it will be important to identify and empower high-quality members of the crowd to make these judgments. Some algorithms already weight responses based on past worker agreement with gold standard data [124, 174], and others are explicitly promoting good workers into management positions [149].

Crowd size has two elements: how many people can participate in the crowd, and how big an active crowd is necessary? Adrenaline has a large source crowd, since anybody with basic aesthetic photography skills can participate. However, it only needs 3–5 members of that crowd at any given time. Friendsourcing applications like Collabio are in the opposite situation: only a very small crowd of people is qualified to participate — the person's social network — and a nontrivial percentage of them need to participate for the system to succeed.

Participation distribution refers to the balance of contributions from members of the crowd. Many natural social computing phenomena are distributed log-normally, so a small number of participants contribute most of the content. Systems can be designed to expect this uneven distribution, or to require a more uniform distribution. For example, traditional web surveys require every respondent to answer every question, while WikiSurveys [171] capture many pairwise votes from dedicated users and use those votes to estimate the preferences for the less active users.

Wherever possible, systems should take advantage of the natural log-normal distribution of participation. For example, on Mechanical Turk, the typical experience is that a small number of workers do many of the tasks. Likewise, requester activity and task completion times naturally follow a log-normal distribution [89, 202]. Requiring a more uniform distribution, for example asking for one thousand respondents for a task, slows down completion time considerably. It is also possible to design directly for this imbalance: for example, friendsourcing incentivizes the small number of very active members of the social network to participate *on behalf of* the less active ones.

Temporality describes how quickly the crowd needs to be recruited and finish its work. Can the task be processed as a batch process and take minutes, hours, days? Or does it need to be completed in realtime?

Decisions about temporarily have a large impact on the crowd size. Smaller crowds are a better match for offline tasks. Chapter 4 describes how a platform could create retainer subscriptions, global retainer pools and global task routing to manage a large realtime crowd, but this still may not approach the size of a slower crowd.

Collaboration has been largely left out of the conversation in crowdsourcing: collaborative systems are more difficult to design and build, and the wisdom of the crowds typically assumes independent judgments [187]. However, letting workers communicate and collaborate can lead to impressive results in collaborative translation [101], and workers produce higher-quality results when they are provided with peer feedback [48]. Should the workers remain completely isolated, or can they work together?

The challenges with collaboration are preventing collusion and social loafing. A preliminary experiment with the retainer model tried embedding a chat room to keep workers interested. However, workers quickly began sharing tricks and information that the requester did not want shared. Social loafing dictates that if a crowd member believes others are participating, they will put out less effort than if they were working alone [119]. However, in many circumstances, direct collaboration may be the best way to get multiple viewpoints heard and evaluated.

Expertise requirements impact the kind of crowd that should be recruited. Many current crowdsourcing tasks can be done by generic, homogenous crowds. However, friendsourcing articulated one space where specialized knowledge is important, and there are many others. The more expertise required, the smaller the potential crowd.

7.2.2 User Design Tradeoffs

Table 7.2 describes the decisions that a system designer now must make with respect to the user. It is critical to these systems that a user remains in control of the

User	
Blocking	Is feedback immediate, or must the user wait for work to complete before the crowd-powered interaction is available?
Initiative	Does the user explicitly request help, or is the crowd actively monitoring for opportunities to step in?
Feedback	Can users tell the crowd whether the work was acceptable? Can the crowd communicate with the users?
Expertise	Does the user have more or less domain expertise than the crowd?

Table 7.2: Design dimensions for the user interaction.

interaction, yet there are many brands of control.

Blocking is parallel to the question of delay: is the user waiting on the crowd-powered results, or can the user continue working while they wait? For example, in Soylent, the user continues to write while the crowd works. In Adrenaline, the user wants to see the picture before they share it or take another one, so the operation is blocking.

Initiative determines whether the user issues commands to the crowd or the crowd takes action automatically as determined by the system. Crowdproof commands are currently issued by the user, but designers could choose to have crowd members continuously monitoring the document and make autonomous recommendations for when to launch Find-Fix-Verify.

Feedback, or visibility of system status, is an important part of any interactive system [151]. The system may show the user the aggregated output, or it might be specific about which crowd workers contribute each element. Likewise, the user may want to communicate task refinements or critiques to the crowd as crowd members work. In Chapter 5, Puppeteer messaged workers as they worked and asked them to adapt their work.

Expertise aims to understand the knowledge imbalance between user and crowd. The user will typically have more context than the crowd. The crowd may have more or less expertise than the user. For example, in Aardvark [84], the question asker knows less about the topic than the crowd who eventually receives the question. In Soylent, however, the user typically has more domain knowledge than the crowd.

7.3 Limitations of Crowdsourcing

To complement systems and user interface research, we also must understand tradeoffs and fundamental limits of crowdsourcing. What are crowds poor at? What can't be solved via crowdsourcing? This section offers some reflections on the limitations of crowd computing.

7.3.1 Limits to Crowd Cognition and Crowd Work

Not every task is a good match for crowdsourcing. Asking crowds for help with the wrong task leads to poor results or no participation. What are the properties of tasks that crowds can complete successfully, and what are the properties of tasks that are out of scope? This section focuses especially on the model of crowdsourcing that parallelizes across a large number of microtasks.

A first limit is what we might call a *startup-contribution ratio.* Consider a ratio: the time that it takes a crowd member to understand and consume any input to complete a task, versus how much time it takes to execute the task the task is understood[4]. For example, consider the task of asking workers to read an entire Wikipedia article before rating its quality [102]. It takes workers a long time to read the article, but very little time to answer the question once the article has been read.

If this startup-contribution ratio is too large, then crowd members cannot quickly get enough context or information to help the requester, and this leads to larger incentives to shirk or cheat. This is roughly comparable to the notion of a streaming algorithm, where the algorithm (worker) has very little working memory available to process the input. For example, crowd members often shirked and didn't read entire articles before trying to submit a task that asked them to do so [102].

A second limit is global consistency or global knowledge. Soylent focuses on very local text edits: it cannot tell an author to cut an entire section or an entire paragraph. While Soylent can guarantee that every word has been read by a crowd worker, no member of the crowd may have enough global knowledge to recommend high-level

[4]This concept developed in conversation with Tim Roughgarden.

action. It is possible to satisfy global constraints through local crowd actions: for example with graph coloring [97] and itinerary planning [207]. However, does a crowd that satisfies global constraints actually have global knowledge? For example, if crowd members read sliding windows on a whole document to generate a summary, does the crowd really have a global knowledge of the document even if no single member does?

A third limit is that, so far, microtask crowds are less well-matched to creative or creation tasks than they are to evaluation tasks. Microtask crowds — especially extrinsically motivated crowds — have become used to tasks with objective outcomes, and tasks which involve quick judgments. Motivational crowding out effects mean that workers are less motivated to work hard on brainstorming or creative tasks, and the current population on Mechanical Turk means that the most creative individuals are probably not even on the platform. Expertise and creativity remain open areas for future work. Yu and Nickerson have structured creative tasks through genetic algorithms to achieve better results [206].

7.3.2 Stifling Individual Abilities

Crowdsourcing is typically framed around input-agreement tasks [198], and this framing systematically biases against individual abilities. If a professional editor were in Soylent's crowd, or if a professional photographer were in Adrenaline's crowd, they would have no ability to sway the system toward an optimal outcome. In fact, the structure of design patterns like Find-Fix-Verify actually restricts individual crowd members from making large or unorthodox changes. Unfortunately, while this practice prevents Eager and Lazy workers from adding errors, it also prevents experts from making beneficial large edits.

More broadly, crowdsourcing techniques tend to bias toward selecting acceptable responses while filtering high-quality and low-quality outliers. Crowdsourcing patterns like Find-Fix-Verify and Rapid Refinement dampen individual responses. For example, Find-Fix-Verify tightly constrains the region where a worker can edit text — whereas a high-quality edit might involve restructuring larger sections of the paragraph. In the future, it will be important to create techniques that recognize high-quality outliers and privilege those workers. Another approach might be to give crowd members the ability to throw an exception and send the input for special processing.

To find a few unusually high-quality solutions, most crowdsourcing sites gather a very large number of submissions. Even if most submissions are low-quality, a small number will be several standard deviations above the mean. For example, competitions such as 99designs[5] and the Netflix Prize[6] succeed when they attract many participants. However, future research can focus on better ways to organize crowds to develop creative solutions.

7.3.3 Scale

This thesis has focused on many parallels that computation shares with crowds, but this comparison breaks down when we consider scale. Crowd computing does not scale as well as silicon. If millions of people suddenly wanted to use Soylent, it might outstrip the capacity of Mechanical Turk.

One reaction to the scale challenge is to draw on optimization algorithms. Statistical techniques have seen early success at minimizing the cost of crowdsourcing workflows [42, 94, 174].

A second response is to recall that, for many applications, the crowd's work can be used to train machine learning systems. As the machine learning results improve, crowds can begin to focus on vetting the algorithm's results, which is often cheaper than doing the work manually. Then, crowds could be reserved for only highly uncertain inputs. Hybrid AI-crowd systems could scale much better than completely crowd-driven systems.

A third reaction is to remember that the notion of "the crowd" will always be a shifting target. If the crowdsourcing research agenda is successful, it will encourage many more individuals to become contractors as part of the crowd. These individuals will bring expertise and better reputations, which will decrease the need for multiple crowd workers performing each task in parallel. Sites like Mechanical Turk are mar-

[5]`http://www.99designs.com`
[6]`http://www.netflixprize.com/`

kets, and increasing wages will also bring more workers into the fold. Systems like Soylent could also shift to a peer exchange model. Users could then edit each others' text to build up credit that they could use in a time of need.

Finally, the scale question becomes less pressing when we consider how powerful crowds are even at their current small scale. A small market like Mechanical Turk has an impressive throughput with perhaps no more than hundreds or thousands of participants online. If crowdsourcing succeeds at helping *hundreds of thousands* of people — both entry-level and expert — to join these platforms as core parts of their careers, these workers could collaborate on impressively large tasks.

7.3.4 Cost

Cost is related to scale. One might argue that systems like Soylent are too expensive to be practical. However, in fact all current document processing tasks also incur significant cost (in terms of computing infrastructure, time, software and salaries); the only difference is that Soylent precisely quantifies the price of each small unit of work. While payment-per-edit may restrict deployment to commercial contexts, it remains an open question whether the gains in productivity for the author are justified by the expense. Furthermore, systems can use crowd outputs to train automatic systems to handle many of the basic edits more automatically.

7.3.5 Privacy

For a task to be crowdsourced, it is shown to a large number of possibly anonymous contributors. This poses clear privacy challenges: crowd members could steal private material and keep it for themselves.

One important first step is to create trusted crowds. For example, these crowd members might sign non-disclosure agreements prior to working for a requester who has privacy concerns. That requester might even build up a private crowd for the purpose. Crowdsourcing platforms could also build up auditing infrastructure to log which crowd members saw a piece of private data.

These questions also suggest new research directions in privacy-preserving crowdsourcing. For example, there may be a formalism we might call *homomorphic crowdsourcing*. In homomorphic crowdsourcing, the goal would be to transform the input so that it is obscured to the crowd worker, but the crowd worker could still perform useful work. Then, platform would need to be able to translate the work back so that it impacts the original input. For example, it may be possible to automatically transform an input image so that the worker cannot see anything private about the image, but they can still (for example) help check face recognition by looking at similarly perturbed other images. The next step in this research would be to define a class of problems where homomorphic crowdsourcing might be possible.

7.3.6 Legal Ownership

Systems such as Soylent also raise questions over legal ownership of the resulting text, which is part-user and part-crowd generated. Do the workers who participate in Find-Fix-Verify gain any legal rights to the document? Today, the answer is no: the Mechanical Turk worker contract explicitly states that it is work-for-hire, so results belong to the requester. Likewise with historical precedent: traditional copyeditors do not own their edits to an article.

If crowds generally do not have legal ownership today, what about tomorrow? Communities such as 4chan and Wikipedia typically take group ownership on their output [19, 55], and crowdsourced product design firms such as Quirky[7] pass on profits to the participants who suggested and developed each design. However, for large companies to feel comfortable using crowds, they need to have intellectual property guarantees similar to employee agreements. These legal norms and codes are likely not yet at the point where everyday employers and employees feel comfortable with crowd work. However, to the extent that the research community can help guide the process, we have the opportunity to make sure neither side is exploited.

[7] `http://www.quirky.com/`

7.3.7 Collusion

In addition to privacy concerns, crowd members could collude or sabotage the requester's work. For example, rogue crowd members sabotaged UC San Diego's entry in the DARPA Shredder Challenge by deliberately introducing errors [159]. Likewise, with Tail Answers, crowd members have the opportunity to directly influence what thousands of web searchers see.

In collusion, several workers agree in advance on the correct answer in order to evade input-agreement filters. Collusion is difficult to prevent, because it is difficult to differentiate between honest independent agreement and several workers collaborating to trick the system. However, it may be possible to automatically transform each worker's input by a known random amount, preventing workers from agreeing on a predefined value in advance [136]. In addition, if the system can control worker-task pairings, randomization can lower the probability that colluding workers wind up on the same task. Games with a Purpose typically take the randomization approach [198].

7.4 Ethics

It is important that the research community ask how crowdsourcing can be a social good, rather than a tool that reinforces inequality. As Stuart Card wrote, "We should be careful to design a world we actually want to live in" [64].

The research literature has begun to come to consensus about the important ethical issues in crowdsourcing. However, problem definitions will not solve the issues, so I will focus on how we might design platforms to drive crowdsourcing toward a positive outcome. Market forces alone will lead crowdsourcing toward lower wages and low-expertise tasks — researchers have the opportunity to adjust this direction and set the agenda.

7.4.1 Wages

Many workers on Mechanical Turk do not make U.S. minimum wage. The estimated hourly wage on Mechanical Turk in 2010 was \$4.80 [89]. Since Mechanical Turk is piecework, there is no guaranteed hourly wage and workers may earn more or less than this amount. Tasks do generate an estimated hourly wage [10], but these numbers are unreliable [169] and requesters may reject the work.

For the platform to be sustainable, workers need to be able to expect a desired living wage. The key change here may be platform guarantees of an *expected wage*: a guarantee of what a worker will make on average if they put forth a good-faith effort. Traditional consultants do not make money every hour, but they likewise can estimate their estimated earnings across a week or a month. Chapter 4 proposed global task routing as a solution to realtime requester needs; it may also be a solution to worker wage needs. In particular, workers should be able to specify a desired wage. Then, depending on the worker's qualifications and task availability, the system can route tasks to that worker in order to guarantee that desired wage in expectation. Some platforms are already pursuing a similar idea: oDesk[8], for example, allows workers to specify a desired wage.

7.4.2 Power Imbalance between Workers and Requesters

Requesters have a large amount of power in the paid crowdsourcing relationship [175, 176, 10, 54]. In Mechanical Turk, requesters can wait days or weeks to pay, and can reject work without reason. Moreover, workers have reputations that can rise and fall, but requesters have no reputations recorded by the platform. Other markets such as oDesk are much more reciprocal. However, it is clear that a small number of requesters can take advantage of many distributed workers.

Projects such as Turkopticon [175] draw workers together to share information about requesters. A formal grievance process might better support workers who are not being paid [10]. However, ultimately, it is in the platform's best interest to act

[8]`http://www.odesk.com`

against poor requesters.

Mechanical Turk has reasons to be biased toward requesters over. For example, requesters are the only ones who are adding money into the system, they are far fewer in number, they have stronger identity requirements and Amazon can monitor them. However, a more healthy balance of power may be positive in the long term.

7.4.3 Crowdsourcing for Evil?

Jonathan Zittrain, a legal scholar at Harvard, has suggested that crowdsourcing could be used for nefarious purposes [29]. As a fictional example, workers may execute a task that appears to be matching faces in photos. However, unbeknownst to them, the government of Iran has listed this task. Iran is using it to identify protesters in photos, then jail them. As a result, crowdsourcing might implicate a large number of unsuspecting workers in goals they might not want to support.

Extrinsic motivations such as money may also provide a cover for ethically ambiguous actions. Today, search engine optimization and content generation tasks (e.g., comments, blog spam) are popular on paid crowdsourcing sites such as Freelancer[9] [100]. This is not surprising: people may lie or take otherwise unscrupulous actions if it benefits them monetarily and they think the other party will not be hurt much [63]. They may feel some amount of ethical dissonance, then seek to reduce it through moral cleansing or comparing their actions to others [7]. Framing things through calculation and money may even suppress negative affective reactions to harmful actions [180, 208].

Our opportunity here is to develop norms and legal codes that keep the interesting and innovative tasks but punish the normatively bad ones. One step in this direction would be to enable collective action and enforce requester transparency in crowdsourcing markets.

[9]`http://www.freelancer.com/`

7.4.4 Cyber-Taylorism vs. Rethinking the Design Process

Crowdsourcing is a renewal for scientific management. Taylorism had positive impacts on optimizing workflows, but it was also seen associated with the dehumanizing elements of factory work and the industrial revolution. Similarly, naive crowdsourcing might treat people as a new kind of API call. This leads both to low-quality systems, as well as unhappy participants.

Involving the user in computing systems led to the development of the human-centered design process. This process makes strong assumptions about the centrality of the user's experience, tasks and goals. Likewise, we need to evolve our design process for crowdsourcing systems to involve the crowds workers' perspective. A theoretical framework such as value-sensitive design may be a good place to start [58].

7.4.5 The Water Cooler for the Crowd: Encouraging Social Interaction

Arguably, doing crowd work today is an isolating experience. Tasks are completed without requiring any communication with the requester or with other workers. While there are forums for workers to share opportunities and frustrations[10], the crowd has no "water cooler".

Encouraging social interaction and collaboration will be an important goal for this work. Peer review on Mechanical Turk increases work quality [48], and collaborative efforts have led to impressive results in poetry translation [101]. However, workplace social interactions have important positive effects on satisfaction, happiness and productivity as well [39, 83]. Social exchange has not yet been designed in as a core element crowdsourcing marketplaces.

[10]`http://www.turkernation.com/`

7.4.6 Career Advancement

Current crowdsourcing platforms tend to focus on generic, non-expert tasks. For example, the ESP Game and Soylent recruit anyone with a basic knowledge of English. These platforms typically do not provide a trajectory for workers to express or develop expertise. This situation results in two challenges: 1) platforms without expert tasks remain at low wages, and 2) platforms without paths to expertise cannot help new workers develop expert skills.

Ideally, the platform should encourage workers to develop skills like programming, visual design, and writing. As work flows from the workers to the requesters, so should expertise flow from the requesters to the workers. One path, as oDesk[11] has tried, are competence tests. A worker could get certified as an intermediate-level Python programmer and thus qualify for new tasks. Better, the platform could provide educational support for a new worker to start as a basic Python scripter and work their way up to become an expert-level programmer. Tasks at different expertise levels could help scaffold the worker along the way.

Crowdsourcing also raises the possibility for new kinds of mini-careers as well. If a requester suddenly needs a crowd of audio mixing experts for the Autotune tool, the platform might deploy incentive schemes so workers recruit other workers [189], then train them up for an afternoon.

[11]`http://www.odesk.com`

Chapter 8

Conclusion

This dissertation introduced *crowd-powered systems*: interactive computing systems that combine machine intelligence with crowd intelligence. This hybrid intelligence enables systems that neither machines nor crowds could support alone: machines may not be able to automate the task yet, and crowds struggle with coordination and quality. The result is mutually beneficial, with computation supporting crowds as they work and crowds guiding the computation.

To conclude, this chapter will review the main contributions of the thesis and consider important avenues for future work.

8.1 Summary of Contributions

The thesis has presented design and implementation patterns for crowd-powered systems that return high-quality results, respond in realtime, depend on personalized knowledge, and mine aggregate crowd behavior. These systems open a design space of deployable applications that draw on collective intelligence and articulate a clear role that computation can play in the wisdom of crowds. They demonstrate how the design of interactive systems can move beyond the traditional tradeoff in user control and system automation (Chapter 1, Figure 8-1) to create hybrid human-computer systems that reach out to the aggregate knowledge, cognition, and perception abilities of many individuals (Figure 8-2).

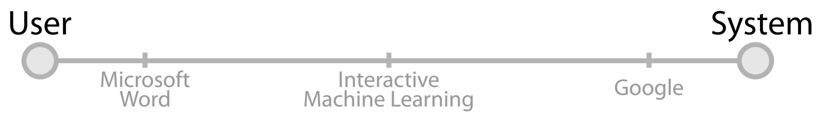


Figure 8-1: As described in Chapter 1, the question of user agency leads to a design axis. At the ends are completely user-controlled interactions and completely system-driven interactions. Designs sometimes split the difference, for example with interactive machine learning [52].

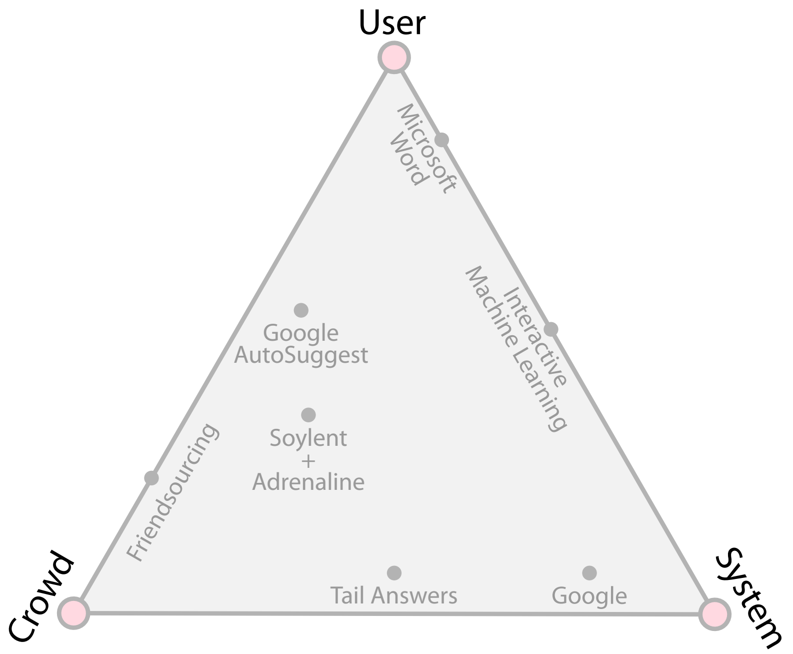


Figure 8-2: Crowd-powered systems add an additional dimension to the design space: crowds may take on tasks that systems cannot perform reliably yet. Soylent relies mostly on crowd contributions, but the system takes initiative in choosing rewrites. There are many under-explored areas of this design space, especially ones that more closely link the crowd to system initiative and artificial intelligence.

Soylent (Chapter 3) opened the design space of crowd-powered systems by demonstrating how crowds could help re-envision canonical interactive applications such as the word processor. It articulated novel interactions like text shortening, support for existing A.I. systems like copyediting and proofreading, and natural language input for macro commands. However, crowd members' wide variation in effort leads to lazy and overeager behavior, resulting in poor-quality results. The *Find-Fix-Verify* design pattern decomposes open-ended problems like text ending into iterative stages that direct workers more carefully and returns higher results. Evaluations tested Soylent across a range of editing tasks: the system found and corrected 82% of grammar errors when combined with automatic checking, shortened text to approximately 85% of its original character length, and executed a variety of human macros successfully.

Interactive systems typically must respond to user input within seconds. Therefore, to create realtime crowd-powered interfaces, we need to dramatically lower crowd latency. *The retainer model* (Chapter 4) pays workers a small wage to wait and respond quickly when asked. Experiments indicate that the retainer model can recruit crowds in two seconds. We then develop a mathematical model of retainer recruitment using queueing theory, which allows requesters to optimize the tradeoff between the probability of a missed task and their cost.

The retainer model opens the door to system designs that depend on realtime crowds. *Adrenaline* (Chapter 5) is a crowd-powered camera where workers quickly filter a short video down to the best single moment for a photo. Unfortunately, even with fast recruitment, work time is slow and photo selection takes longer than users are willing to wait. *Rapid refinement* observes early signs of agreement in synchronous crowds and dynamically narrows the search space to focus on promising directions. This approach produces results that, on average, are of more reliable quality and arrive faster than the fastest crowd member working alone.

Soylent and Adrenaline demonstrate the power of generic crowd intelligence, but many applications require knowledge that generic crowds might not know. First, *friendsourcing* (Chapter 6) collects accurate information available only to a small, socially-connected group of individuals. Social friend tagging *(Collabio)* and news-sharing *(FeedMe)* applications produce accurate information about individuals and augment data that could have been found on Facebook or the Web. This social data supports personalized applications such as question-routing and recommender systems. Second, by aggregating crowd data, systems ease their dependence on live crowds and enable support for a large number of less common user goals. *Tail Answers* aggregate activity traces from web searchers to directly respond to a large number of long-tail information needs. This approach significantly improves users' subjective ratings of search quality and their ability to solve needs without clicking through to a result.

Broadly, this thesis makes contributions in the areas of design, crowd computing, and social computing:

1. **Design.** Crowd-powered systems enable a new class of applications that give end users direct access to high-level commands and natural interaction.

2. **Crowd computing.** To complete open-ended tasks like text shortening or proofreading, decomposition design patterns like Find-Fix-Verify will guide workers toward high-quality results. To enable realtime crowdsourcing, a combination of retainer recruitment and synchronous crowd algorithms produce crowds in seconds and results soon after.

3. **Social computing.** Crowd computing structures the interactions between participants on the web to help them accomplish complex tasks. Designing new social interactions like friendsourcing can create crowds to collect information that existing crowds might not know.

8.2 Impact and Recent Developments

While Soylent was one of the first crowd-powered systems, the broader research community has articulated many more systems in the 18 months since the work was published.

These systems draw crowds into many new domains. Some push on prosocial goals: PlateMate crowdsources calorie counts using photographs of meals [152], while VizWiz helps blind users ask questions about their environments [21]. Others rethink standard information-centered activities such as task planning [207], search [23], translation [4], and authoring maps [185]. Database researchers have articulated the power of using crowds to relax closed-world assumptions in databases [56, 137, 157]. Crowds are also tied into existing fields by extending the reach of robotics [182, 117], design [206], machine vision [165, 205] and graphics [62].

Find-Fix-Verify has been directly adapted by researchers for tasks like image segmentation [152], map labeling [185], and formal crowd programming languages [145].

8.3 Future Work

Crowd computing is a nascent field. It has attracted interest across subdisciplines of computer science as well as social science. It has established its own conferences[1] and made major inroads at traditional computer science venues such as CHI, UIST, CSCW, SIGMOD, VLDB, and AAAI.

It is time to begin defining long-term goals for crowd computing. This section articulates several such millennium goals for crowdsourcing and crowd-powered systems.

8.3.1 Hybrid Crowd–A.I. Systems

The systems presented in this thesis do not deeply integrate with artificial intelligence. However, machine learning and artificial intelligence complement crowdsourcing naturally.

One view is to create systems that dynamically trade off crowd and machine intelligence. Such as system would rely heavily on crowd data early, to train the machine learning algorithm, then phase out the crowd as the algorithm improves.

[1]AAAI HCOMP (Human Computation) and Collective Intelligence

Eventually the crowd could be used only to vet highly uncertain inputs. Similarly, crowd members could be cast as stump learners in an ensemble learner, where a meta-algorithm learns when to trust crowd members and when to trust the machine learning algorithm.

Rather than train existing machine learning algorithms with crowd data, it might be possible to design machine learning algorithms directly for crowds. Humans have biases when they label data, but this bias can be modeled and compensated for. In a semi-supervised setup, the algorithm may want to balance between expensive, slow human labels and uncertain labels the algorithm assigns. These algorithms should aim to more precisely model the tradeoffs between machine intelligence and crowd intelligence.

8.3.2 Crowdsourcing Markets

While Amazon Mechanical Turk is a useful prototyping platform, its current incarnation has serious limitations. Researchers have the opportunity to define what that platform should look like.

It will be critical to move toward more expert work. For example, platforms that could assemble flash crowds of experts to pipeline large tasks — companies and organizations that assemble, work together for an afternoon, then disperse. Starting with a sketch for a user interface, such a platform could find a designer to create a mock-up, pass the mock-up to a usability professional for testing, loop until the design meets usability goals a few hours later, and finally recruit a programmer to implement the interface. Put another way: what would it take to crowdsource a presentation, or an entire software program, or a symphony?

Collaborations with economics, policy and legal scholars need to define norms and incentives for honest work and payments. Right now, Mechanical Turk is a market for lemons [90]: workers cannot trust requesters to pay, so they do poor work, and requesters cannot trust workers, so they need to hire multiple workers per task and pay less. Computer science and economics researchers have developed auction designs, prediction markets, and other mechanisms for honest reporting. In order to build complex systems, the basic market mechanisms need to be reliable.

To put crowdsourcing in the hands of many more end users, the authoring and request toolkits need to become much more mature. While end users commonly organize human teams around them to complete tasks, crowdsourcing tools do not yet let them do this with crowds. Many paradigms exist: TurKit presents an interative programming interface to Mechanical Turk [129], CrowdWeaver offers a dataflow visualization of pipelined workflows [103], and Turkomatic has a single request box similar to Google [114]. These tools need to give end-users the ability to direct a workflow, monitor and debug it. Furthermore, end users need to produce usable interfaces for crowd members without gaining expertise in the user-centered design process.

8.3.3 A Science of Crowdsourcing

There has been explosive growth in crowdsourcing over the past two years, but we need to pair discovery with principles. Specifically, we need to develop design patterns and best practices for crowd computing. We also lack methods to analyze and compare the complexity of crowd computing algorithms.

First, crowdsourcing needs a core literature in design patterns (e.g., [59]). Patterns like Find-Fix-Verify, Iterate-and-Vote [128], and Price-Divide-Solve [114] all help enable crowd-powered systems; next, we must generalize, look critically at these patterns, and understand their strengths as well as their weaknesses.

Second, a formal framework would allow us to compare approaches along axes such as cost, crowd size, latency, quality and worker stress. How can we formalize the ways in which one crowd algorithm is better than another? Runtime analysis similar to Big O notation is a good starting point, because runtime typically directly impacts costs and latency [113, 156]. However, it is not enough to only consider runtime: issues like mental workload (measurable via NASA TLX [75]), work quality, and the startup-contribute ratio from Chapter 7 all could to be modeled.

Third, where formal analysis might not succeed, the field should settle on benchmarks: public datasets and tasks to optimize. Tasks like handwriting transcription [128], text shortening and image labeling [199] are clear candidates. By centering in on hard problems, the field can move from a problem-setting enterprise to a problem-solving enterprise and seek measurable progress.

8.3.4 Autonomous, Self-Correcting Crowds

The crowd-powered systems in this thesis are all ultimately under the control of a single user. This is a useful constraint, because the user can specify the goals of the system and adjust course as needed.

However, it will be important to pursue a goal of self-organizing, self-correcting crowds. Such crowds should be able to start from a high-level goal such as "write an encyclopedia article" and then organize the workflow decomposition and team structure to accomplish the task [104, 114]. Moreover, these crowds will need to recognize when they are in a local optimum or have strayed from the original goal, then create a plan to achieve it.

Crowd memory will be critical to these long-running systems. A crowdsourced personal assistant needs to act as if it remembers every past interaction with the user, even though the individuals who originally experienced that interaction are no longer present. Likewise, as long-running processes continue, crowds need to hand off local knowledge to new members [118].

8.3.5 Large-Scale Systems

The crowd-powered systems in this thesis support very focused, local goals such as text shortening. What if Soylent wanted to shorten an entire paper, or if Adrenaline wanted to sort through an entire vacation's worth of photos? These applications would require much larger-scale systems, where design patterns like Find-Fix-Verify become primitives. These goals would push on a set of challenges with scale, reliability, and propagated error in crowd computing systems.

8.4 Looking Ahead

The fundamental idea of this thesis is to tightly bind crowd intelligence to interaction, to software, and to computation. As a result, the user interface for the crowd participants becomes a core part of software design. Rather than isolate and separate the user interface from system implementation, this thesis involves human contributions as first-class elements of software. The resulting systems are more expressive and powerful than traditional interactive software, and the resulting crowds succeed at tasks that traditional crowds do not. We believe that this tight integration of user, crowd, and system will be a powerful model for interactive computing systems.

Appendix A

Soylent Evaluation Texts

This section contains the full input texts from the Soylent evaluations.

A.1 Shortn inputs

These texts were provided as inputs to the Shortn evaluation in Section 3.4.1.

A.1.1 Blog

Print publishers are in a tizzy over Apple's new iPad because they hope to finally be able to charge for their digital editions. But in order to get people to pay for their magazine and newspaper apps, they are going to have to offer something different that readers cannot get at the newsstand or on the open Web. We've already seen plenty of prototypes from magazine publishers which include interactive graphics, photo slide shows, and embedded videos.

But what should a magazine cover look like on the iPad? After all, the cover is still the gateway to the magazine. Theoretically, it will still be the first page people see, giving them hints of what's inside and enticing them to dive into the issue. One way these covers could change is that instead of simply repurposing the static photographs from the print edition, the background image itself could be some sort of video loop. Jesse Rosten, a photographer in California, created the video mockup below of what a cover of Sunset Magazine might look like on the iPad (see video below).

The video shows ocean waves gently lapping a beach as the title of the magazine and other typographical elements appear on the page almost like movie credits. He points out that these kinds of videos will have to be shot in a vertical orientation rather than a horizontal landscape one. This is just a mockup Rosten came up with on his own, but the designers of these new magazine apps should take note. The only way people are going to pay for these apps is if they create new experiences for readers.

A.1.2 Classic UIST Paper [92]

The graphical user interface (GUI) has proven both a successful and durable model for human-computer interaction which has dominated the last decade of interface design. At the same time, the GUI approach falls short in many respects, particularly in embracing the rich interface modalities between people and the physical environments they inhabit. Systems exploring augmented reality and ubiquitous computing have begun to address this challenge. However, these efforts have often taken the form of exporting the GUI paradigm to more world-situated devices, falling short of much of the richness of physical-space interaction they seek to augment.

In this paper, we present research developing "Tangible User Interfaces" (TUIs) — user interfaces employing physical objects, instruments, surfaces, and spaces as physical interfaces to digital information. In particular, we present the metaDESK system, a graphically intensive system driven by interaction with graspable physical objects. In addition, we introduce a prototype application driving an interaction with geographical space, Tangible Geospace, to illustrate our approach.

The metaDESK effort is part of the larger Tangible Bits project. The Tangible Bits vision paper introduced the metaDESK along with two companion platforms, the transBOARD and ambientROOM. Together, these platforms explore both graspable physical objects and ambient environmental displays as means for seamlessly coupling people, digital information, and the physical environment.

The metaDESK system consists of several components: the desk, a nearly-horizontal backprojected graphical surface; the active lens, an arm-mounted flat-panel display; the passive lens, an optically transparent surface through which the desk projects; and an assortment of physical objects and instruments which are used on desk's surface. These components are sensed by an array of optical, mechanical, and electromagnetic field sensors.

Our research with the metaDESK system focuses on the use of tangible objects — real physical entities which can be touched and grasped — as driving elements of human-computer interaction. In particular, we are interested in pushing back from the GUI into the real world, physically instantiating many of the metaphorical devices the GUI has popularized. Simultaneously, we have attempted to push forward from the unaugmented physical world, inheriting from the richness of various historical instruments and devices often "obsoleted" by the advent of the computer.

In addition, we more broadly explore the use of physical affordances within TUI design. For example, our active lens is not only grounded in the metaphor of a jeweler's magnifying lens; it also looks, acts, and is manipulated like such a device. In this way, the active lens has a certain legibility of interface in that its affordances suggest and support user's natural expectations from the device.

In the following sections, we present our design approach towards making user interfaces tangible. The operating scenario of the Tangible Geospace prototype is then presented. This is followed by a description of the metaDESK implementation, including display, sensor, and software architectures. Interaction issues encountered with the prototype are then discussed, followed by future work and conclusions.

A.1.3 Draft UIST Paper [194]

Too often, even the best information retrieval tools cannot help us find what we are seeking, because the information we want was never entered. This can happen for many reasons. Sometimes, we simply do not recognize that the information might be needed later. At other times, the perceived cost to launch and navigate through multiple applications to capture the information seems too high for the currently perceived value of the information. Lastly, our strong desire to record some information can be stymied by the fact that there is no natural place for it—no place where we have confidence that we will be able to find it when we need it, or, similarly, no native application that may be associated with the particular kind of data being entered.

Many of these problems vanish if we turn to a much older recording technology—text. Recording a fragment of text simply requires picking up a pen or typing at a keyboard. When we enter text, each (pen or key) stroke is being used to record the actual information we care about—none is wasted on application navigation or configuration. The linear structure of text means there's always an obvious place to put anything—at the end. And the free form of text means we can record anything we want to about anything, without worrying whether it fits some application schema or should be split over multiple applications. All of this means that we have to do less to record text, which makes it more efficient and also less of an interruption and distraction than using a complex application.

While text is an outstanding solution for recording information, its weakness lies in retrieval. Text's fixed linear form reduces us to scanning through it for information we need. Even with electronic text, the lack of structure means we cannot filter or sort by various properties of the information. When [1] we aren't sure what we want, a blank text search box offers few cues to help us construct an appropriate query. The [2] shorthand we use to record information in a given context can make it incomprehensible when we return to it later without that context. And only the text we explicitly enter is recorded, without any of the related information that might be known to a sophisticated application.

In this paper we argue that it is possible and desirable to combine the easy input affordances of text with the powerful retrieval and visualization capabilities of graphical applications. We present WenSo, a tool that uses lightweight text input to capture richly structured information for later retrieval and navigation in a graphical environment. WenSo provides

- entry of information by typing arbitrary scraps of text (with all the text-input benefits mentioned above)

- inclusion of structured information in the text through a natural and extensible "pidgin"
- extraction of structure through lightweight recognition of entities and relationships between them
- association of automatically-measured context with the information being recorded
- search and faceted browsing based on tags, entities, and relations for finding relevant text scraps
- automatic routing of relevant pieces of the entered information to structured applications such as calendar, address book, and web browser so that it can be retrieved and visualized using those domain-specific tools.

In order to deliver these interactions, we had to solve several key problems: capturing structure from text not entered in a form, modeling capture of desktop state for appropriate association with a scrap, and integration of captured data for use with existing applications. In the following sections we present the related work that informs our approach, describe the interaction design and describe in our solutions for the key system implementation challenges. We then discuss the future research opportunities both for extending the WenSo platform, but most immediately, for using the platform to determine what happens in terms of information scrap entry and reuse behaviour once these new affordances have been provided.

A.1.4 Rambling Enron E-mail

Lyne, My name is Mark Bain and I'm the Web Site Adminstrator for the Mariners. I would be glad to help out in any way. Please pass this email and my phone numbers (281-379-6896 hm. 281-518-3251 wk.) to your website person. I put alot of info in this email to help get started but if it is a little overwhelming, just call.

A previous board member, Steve Burleigh, created our web site last year and gave me alot of ideas. For this year, I found a web site called eTeamZ that hosts web sites

for sports groups. Check out our new page:
http://www.eteamz.com/swimmariners/

eTeamsZ is really easy to use and does not require much web site knowledge. They do the formatting and you supply the info. If your web guru wants to get creative, they can do some custome stuff as well. The best part, however, is that it is FREE. Of course with free you have to put up with a little advertising but since this site is tailor made for sports (meetschedules, practice schedules calendars, news, maps, etc.), I think it is worth it.

If you don't decide to use eTeamZ, that's fine too. I've created some other web sites and should be able to answer some questions. Sounds like you have some experience so it should be no problem. Good luck and let me know what else you need.

eTeamZ TIPS:
Oh, here are some tips on getting started with eTeamZ. (difficulties that I had anyway):
Registration- http://www.eteamz.com/company/sites/register/
The toughest part was coming up with a nickname and a user name for the websites. Seems that most of the users are into baseball and Mariners was taken. I think I used Mariners Swim Team for nickname and goMariners for the username because I kept hitting other name that were being used.

WebSite- When I got to the "build you web" page, I chose Team Web Site. They also had options for Leagues and Orgs. My best advice is to start populating things via the admin page and bring up another windows to check the progress of what is actually displayed. PLUS — They have a premium service called PLUS that you pay for but I didn't see much use for it except you get rid of some of the advertising banners. They also show that you can use this site for registration and other things but we are keeping it pretty simple.

A.1.5 Technical Writing [5]

FAWN-DS uses an in-memory (DRAM) Hash Index to map 160-bit keys to a value stored in the Data Log. It stores only a fragment of the actual key in memory to find a location in the log; it then reads the full key (and the value) from the log and verifies that the key it read was, in fact, the correct key. This design trades a small and configurable chance of requiring two reads from flash (we set it to roughly 1 in 32,768 accesses) for drastically reduced memory requirements (only six bytes of DRAM per key-value pair).

Figure 3 shows the pseudocode that implements this design for Lookup. FAWN-DS extracts two fields from the 160-bit key: the i low order bits of the key (the index bits) and the next 15 low order bits (the key fragment). FAWN-DS uses the index bits to select a bucket from the Hash Index, which contains 2i hash buckets. Each bucket is only six bytes: a 15-bit key fragment, a valid bit, and a 4-byte pointer to the location in the Data Log where the full entry is stored.

Lookup proceeds, then, by locating a bucket using the index bitsand comparing the key against the key fragment. If the fragments do not match, FAWN-DS uses hash chaining to continue searching the hash table. Once it finds a matching key fragment, FAWN-DS reads the record off of the flash. If the stored full key in the on-flash record matches the desired lookup key, the operation is complete. Otherwise, FAWN-DS resumes its hash chaining search of the inmemory hash table and searches additional records. With the 15-bit key fragment, only 1 in 32,768 retrievals from the flash will be incorrect and require fetching an additional record.

A.2 Crowdproof inputs

These texts were provided as inputs to the Shortn evaluation in Section 3.4.2.

A.2.1 Passes Word's Checker

Marketing are bad for brand big and small. You Know What I am Saying. It is no wondering that advertisings are bad for company in America, Chicago and Germany. Updating of brand image are bad for processes in one company and many companies.[1]

[1]From `http://faculty.washington.edu/sandeep/check`

A.2.2 English as a Second Language

However, while GUI made using computers be more intuitive and easier to learn, it didn't let people be able to control computers efficiently. Masses only can use the software developed by software companies, unless they know how to write programs. In other words, if one who knows nothing about programming needs to click through 100 buttons to complete her job everyday, the only thing she can do is simply to click through those buttons by hand every time. But if she happens to be a computer programmer, there is a little chance that she can write a program to automate everything. Why is there only a little chance? In fact, each GUI application is a big black box, which usually have no outward interfaces for connecting to other programs. In other words, this truth builds a great wall between each GUI application so that people have difficulty in using computers efficiently. People still do much tedious and repetitive work in front of a computer.

A.2.3 Notes from a Talk: NoSQL in the Cloud

Blah blah blah-argument about whether there should be a standard "nosql storage" API to protect developers storing their stuff in proprietary services in the cloud. Probably unrealistic. To protect yourself, use an open software offering, and self-host or go with hosting solution that uses open offering.

Interesting discussion on disaster recovery. Since you've outsourced operations to the cloud, should you just trust the provider w/ diaster recovery. People kept talking about busses driving through datacenters or fires happening. What about the simpler problem: a developer drops your entire DB. Need to protect w/ backups no matter where you host.

A.2.4 Bad Wikipedia Page

Dandu Monara (Flying Peacock, Wooden Peacock), The Flying machine able to fly. The King Ravana (Sri Lanka) built it. Accorinding to hindu believes in Ramayanaya King Ravana used "Dandu Monara" for abduct queen Seetha from Rama. According to believes ”Dandu Monara” landed at Werangatota, about 10 km from Mahiyangana. It is the hill station of Nuwara Eliya in central Sri Lanka.

A.2.5 Draft UIST Paper [194]

Same as Section A.1.3.

Bibliography

[1] Mark S Ackerman and Thomas W Malone. Answer Garden: a tool for growing organizational memory. In *Proc. GROUP '90*, 1990.

[2] Eugene Agichtein, Steve Lawrence, and Luis Gravano. Learning to find answers to questions on the Web. *ACM TOIS*, 4(2):129–162, May 2004.

[3] Salman Ahmad, Alexis Battle, Zahan Malkani, and Sepander Kamvar. The jabberwocky programming environment for structured social computing. In *Proc. UIST '11*, October 2011.

[4] Vamshiti Amba, Stephan Vogel, and Jaime Carbonell. Collaborative workflow for crowdsourcing translation. In *Proc. CSCW '12*. ACM, 2012.

[5] David G. Andersen, Jason Franklin, Michael Kaminsky, Amar Phanishayee, Lawrence Tan, and Vijay Vasudevan. FAWN: a fast array of wimpy nodes. In *Proc. SOSP '09*, 2009.

[6] Judd Antin and Aaron Shaw. Social desirability bias and self-reports of motivation: a study of amazon mechanical turk in the US and India. In *Proc. CHI '12*, May 2012.

[7] Shahar Ayal and Francesca Gino. Honest rationales for dishonest behavior. *The Social Psychology of Morality: Exploring the Causes of Good and Evil*, 2011.

[8] Michele Banko, Michael J Cafarella, Stephen Soderland, Matt Broadhead, and Oren Etzioni. Open information extraction for the web. In *Proc. IJCAI '07*, 2007.

[9] Daniel W. Barowy, Emery D. Berger, and Andrew McGregor. AUTOMAN: A Platform for Integrating Human-Based and Digital Computation. Technical report, UM-CS-2011-044, University of Massachusetts, Amherst, 2012., 2012.

[10] Benjamin B. Bederson and Alexander J. Quinn. Web workers unite! addressing challenges of online laborers. In *Extended Abstracts CHI '11*, May 2011.

[11] Gerard Beenen, Kimberly Ling, Xiaoqing Wang, Klarissa Chang, Dan Frankowski, Paul Resnick, and Robert E. Kraut. Using social psychology to motivate contributions to online communities. In *Proc. CSCW '04*, 2004.

[12] Adam Bernstein. The Acknowledged Master of the Moment. *Washington Post*, August 2004.

[13] Michael Bernstein, Adam Marcus, David R Karger, and Robert C Miller. Enhancing Directed Content Sharing on the Web. In *Proc. CHI '10*, 2010.

[14] Michael Bernstein, Desney Tan, Greg Smith, Mary Czerwinski, and Eric Horvitz. Collabio: A Game for Annotating People within Social Networks. In *Proc. UIST '09*, August 2009.

[15] Michael Bernstein, Desney Tan, Greg Smith, Mary Czerwinski, and Eric Horvitz. Personalization via Friendsourcing. *ACM Transactions on Human-Computer Interaction (TOCHI)*, 17(2):1–28, 2010.

[16] Michael S Bernstein, Joel Brandt, Robert C Miller, and David R Karger. Crowds in two seconds: Enabling realtime crowd-powered interfaces. In *Proc. UIST '11*, 2011.

[17] Michael S Bernstein, David R Karger, Robert C Miller, and Joel Brandt. Analytic Methods for Optimizing Realtime Crowdsourcing. *Proc. Collective Intelligence 2012*, 2012.

[18] Michael S Bernstein, Greg Little, Robert C Miller, Björn Hartmann, Mark S Ackerman, David Crowell, and Katrina Panovich. Soylent: A Word Processor with a Crowd Inside. In *Proc. UIST '10*, 2010.

[19] Michael S Bernstein, Andrés Monroy-Hernandez, Drew Harry, Paul Andre, Katrina Panovich, and Greg Vargas. 4chan and /b/: An Analysis of Anonymity and Ephemerality in a Large Online Community, 2011.

[20] Michael S Bernstein, Jaime Teevan, Susan Dumais, Daniel Liebling, and Eric Horvitz. Direct answers for search queries in the long tail. In *Proc. CHI '12*, CHI '12, 2012.

[21] Jeffrey P. Bigham, Chandrika Jayant, Hanjie Ji, Greg Little, Andrew Miller, Robert C. Miller, Robin Miller, Aubrey Tatrowicz, Brandyn White, Samuel White, and Tom Yeh. VizWiz: Nearly Real-time Answers to Visual Questions. In *Proc. UIST '10*, 2010.

[22] Daniel Billsus, Michael J. Pazzani, and James Chen. A learning agent for wireless news access. In *Proc. IUI '00*, 2000.

[23] Alessandro Bozzon, Marco Brambilla, and Stefano Ceri. Answering search queries with CrowdSearcher. In *Proc. WWW '12*, WWW '12, 2012.

[24] Steve Branson, Catherine Wah, Florian Schroff, Boris Babenko, Peter Welinder, Pietro Perona, and Serge Belongie. Visual recognition with humans in the loop. *Proc. ECCV 2010*, 2010.

[25] Eric Brill, Susan Dumais, and Michele Banko. An analysis of the AskMSR question-answering system. In *Proc. EMNLP '02*, volume 10, July 2002.

[26] Andrei Broder. A taxonomy of web search. *ACM SIGIR Forum*, 36(2):3, September 2002.

[27] Peter Brusilovsky, Alfred Kobsa, and Wolfgang Nejdl, editors. *The Adaptive Web*. Springer-Verlag, 2007.

[28] Chris Callison-Burch and Mark Dredze. Creating Speech and Language Data With Amazon's Mechanical Turk. In *Proc. NAACL HLT 2010 Workshop on Creating Speech and Language Data with Amazon's Mechanical Turk*, June 2010.

[29] MacGregor Campbell. *The sinister powers of crowdsourcing.* http://www.newscientist.com/article/dn18315-innovation-the-sinister-powers-of-crowdsourcing.html, 2009.

[30] Stuart K Card, Thomas P Moran, and Allen Newell. *The psychology of human-computer interaction.* Lawrence Erlbaum, 1983.

[31] Tao Chen, Ming-Ming Cheng, Ping Tan, Ariel Shamir, and Shi-Min Hu. Sketch2Photo. *ACM Transactions on Graphics*, 28(5), 2009.

[32] Lydia Chilton, John Horton, Robert C. Miller, and Shiri Azenkot. Task search in a human computation market. In *Proc. HCOMP '10*, 2010.

[33] Lydia Chilton and Jaime Teevan. Addressing people's information needs directly in a web search result page. In *Proc. WWW '11*, 2011.

[34] James Clarke and Mirella Lapata. Models for sentence compression: a comparison across domains, training requirements and evaluation measures. In *Proc. ACL '06*, 2006.

[35] Michael F. Cohen and Richard Szeliski. The moment camera. *Computer*, 39(8):40–45, 2006.

[36] Trevor Cohn and Mirella Lapata. Sentence compression beyond word deletion. *Proc. COLING '08*, 2008.

[37] Seth Cooper, Firas Khatib, Adrien Treuille, Janos Barbero, Jeehyung Lee, Michael Beenen, Andrew Leaver-Fay, David Baker, Zoran Popović, and Foldit-Players. Predicting protein structures with a multiplayer online game. *Nature*, 466(7307):756–760, 2010.

[38] Justin Cranshaw and Aniket Kittur. The polymath project: lessons from a successful online collaboration in mathematics. In *Proc. CHI '11.* ACM, 2011.

[39] Russell Cropanzano and Marie S. Mitchell. Social Exchange Theory: An Interdisciplinary Review. *Journal of Management*, 31(6):874–900, December 2005.

[40] Edward Cutrell and Zhiwei Guan. What are you looking for?: an eye-tracking study of information usage in web search. In *Proc. CHI '07*, 2007.

[41] Allen Cypher. *Watch What I Do.* MIT Press, Cambridge, MA, 1993.

[42] Peng Dai, Mausam, and Dan Weld. Decision-theoretic control of crowd-sourced workflows. In *Proc. AAAI '10*, 2010.

[43] Peng Dai, Mausam, and Dan Weld. Artificial Intelligence for Artificial Artificial Intelligence. In *Proc. AAAI '11*, 2011.

[44] A P Dawid and A M Skene. Maximum likelihood estimation of observer error-rates using the EM algorithm. *Applied Statistics*, pages 20–28, 1979.

[45] Ofer Dekel and Ohad Shamir. Vox populi: Collecting high-quality labels from a crowd. *Proc. 22nd Annual Conference on Learning Theory*, 2009.

[46] Joan Morris DiMicco and David R. Millen. Identity management: multiple presentations of self in facebook. In *Proc. GROUP '07*, 2007.

[47] Paul Dourish and Victoria Bellotti. Awareness and coordination in shared workspaces. In *Proc. CSCW '92*, 1992.

[48] Steven Dow, Anand Kulkarni, Scott Klemmer, and Björn Hartmann. Shepherding the crowd yields better work. In *Proc. CSCW '12*, February 2012.

[49] Casey Dugan, Michael Muller, David R. Millen, Werner Geyer, Beth Brownholtz, and Marty Moore. The dogear game: a social bookmark recommender system. In *Proc. GROUP '07*, 2007.

[50] Douglas C Engelbart and William K English. A research center for augmenting human intellect. In *Proc. Fall Joint Computer Conference, Part I.* ACM, 1968.

[51] Brynn M Evans and Ed H Chi. Towards a model of understanding social search. In *Proc. CSCW '08.* ACM, 2008.

[52] Jerry Alan Fails and Dan R. Olsen. Interactive machine learning. In *Proc. IUI '03*. ACM, 2003.

[53] Stephen Farrell, Tessa Lau, Stefan Nusser, Eric Wilcox, and Michael Muller. Socially augmenting employee profiles with people-tagging. In *Proc. UIST '07*, 2007.

[54] Karën Fort, Gilles Adda, and K. Bretonnel Cohen. Amazon mechanical turk: Gold mine or coal mine? *Computational Linguistics*, 37(2):413–420, 2011.

[55] Andrea Forte and Amy Bruckman. Why do people write for wikipedia? Incentives to contribute to open-content publishing. *Proc. of GROUP '05*, 5, 2005.

[56] Michael Franklin, Donald Kossmann, Tim Kraska, Sukriti Ramesh, and Reynold Xin. CrowdDB: answering queries with crowdsourcing. *Proc. SIGMOD '11*, 2011.

[57] Bruno S Frey and Felix Oberholzer-Gee. The cost of price incentives: An empirical analysis of motivation crowding-out. *The American Economic Review*, 87(4):746–755, 1997.

[58] Batya Friedman. Value-sensitive design. *interactions*, 3(6):16–23, 1996.

[59] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design patterns: elements of reusable object-oriented software*. Addison-Wesley Professional, 1995.

[60] Tomio Geron. Why Start-Ups Must Pay Attention To Whats Behind The Curtain. In *Wall Street Journal*. http://blogs.wsj.com/venturecapital/2010/04/24/how-a-start-up-grew-by-paying-attention-to-whats-behind-the-curtain/, 2010.

[61] Eric Gilbert and Karrie Karahalios. Predicting tie strength with social media. In *Proc. CHI '09*, 2009.

[62] Yotam Gingold, Ariel Shamir, and Daniel Cohen-Or. Micro Perceptual Human Computation. *To appear in ACM Transactions on Graphics (TOG)*, 2012.

[63] Uri Gneezy. Deception: The role of consequences. *The American Economic Review*, 95(1):384–394, 2005.

[64] Richard Gold. *The plenitude: creativity, innovation, and making stuff.* The MIT Press, 2007.

[65] Mark Granovetter. The Strength of Weak Ties. *American Journal of Sociology*, 78(6):1360–1380, 1973.

[66] Saul Greenberg and Ralph Bohnet. GroupSketch: A multi-user sketchpad for geographically-distributed small groups. In *Proc. Graphics Interface'91*, 1991.

[67] David Alan Grier. *When Computers Were Human.* Princeton University Press, 2005.

[68] David Alan Grier. Error Identification and Correction in Human Computation: Lessons from the WPA. In *Proc. HCOMP '11*, 2011.

[69] Donald Gross and Carl Harris. Fundamentals of queueing theory. 1998.

[70] Stephen Guo, Aditya Parameswaran, and Hector Garcia-Molina. So Who Won? Dynamic Max Discovery with the Crowd. Technical report, Stanford University, November 2011.

[71] Aakar Gupta, William Thies, Edward Cutrell, and Ravin Balakrishnan. mClerk: Enabling Mobile Crowdsourcing in Developing Regions. In *Proc. CHI '12*, 2012.

[72] Severin Hacker and Luis Von Ahn. Matchin: eliciting user preferences with an online game. In *Proc. CHI '09.* ACM, 2009.

[73] Severin Hacker and Luis von Ahn. *Duolingo.* www.duolingo.com, 2012.

[74] F. Maxwell Harper, Sherry Xin Li, Yan Chen, and Joseph A. Konstan. Social Comparisons to Motivate Contributions to an Online Community. In *Proc.*

Persuasive Technology '07, volume 4744 of *Lecture Notes in Computer Science*, 2007.

[75] Sandra G Hart and Lowell E Staveland. Development of NASA-TLX (Task Load Index): Results of empirical and theoretical research. *Human mental workload*, 1:139–183, 1988.

[76] Bjoern Hartmann and Folks on-the Internet. *Amazing but True Cat Stories.* 2008.

[77] Björn Hartmann, Daniel MacDougall, Joel Brandt, and Scott R Klemmer. What Would Other Programmers Do? Suggesting Solutions to Error Messages. In *Proc. CHI '10*, 2010.

[78] Marti A. Hearst. *Search User Interfaces.* Cambridge University Press, 2009.

[79] Jeffrey Heer and Michael Bostock. Crowdsourcing Graphical Perception: Using Mechanical Turk to Assess Visualization Design. In *Proc. CHI '10*, 2010.

[80] Kurtis Heimerl, Brian Gawalt, Kuang Chen, Tapan S. Parikh, and Bjoern Hartmann. Communitysourcing: Engaging Local Crowds to Perform Expert Work Via Physical Kiosks. In *Proc. CHI '12*, 2012.

[81] Joseph M Hellerstein and David L Tennenhouse. Searching for Jim Gray: a technical overview. *Communcations of the ACM*, 54(7):77–87, July 2011.

[82] Raphael Hoffmann, Saleema Amershi, Kayur Patel, Fei Wu, James Fogarty, and Daniel S. Weld. Amplifying community content creation with mixed initiative information extraction. In *Proc. CHI '09*, 2009.

[83] Jim Hollan and Scott Stornetta. Beyond being there. In *Proc. CHI '92*, 1992.

[84] Damon Horowitz and Sepandar D Kamvar. The anatomy of a large-scale social search engine. In *Proc. WWW '10*, 2010.

[85] Jeff Howe. The rise of crowdsourcing. *Wired magazine*, 14(6), 2006.

[86] Jeff Howe. *Crowdsourcing: How the power of the crowd is driving the future of business*. Century, 2008.

[87] Takeo Igarashi, Tomer Moscovich, and John F Hughes. As-rigid-as-possible shape manipulation. *ACM Transactions on Graphics (TOG)*, 24(3):1134–1141, 2005.

[88] Panagiotis G. Ipeirotis, Foster Provost, and Jing Wang. Quality management on amazon mechanical turk. In *Proc. HCOMP '10*. ACM, 2010.

[89] Panos G Ipeirotis. Analyzing the Amazon Mechanical Turk marketplace. *XRDS: Crossroads, The ACM Magazine for Students*, 17(2):16–21, 2010.

[90] Panos G Ipeirotis. *Mechanical Turk, Low Wages, and the Market for Lemons*. http://www.behind-the-enemy-lines.com/2010/07/mechanical-turk-low-wages-and-market.html, 2010.

[91] Hiroshi Ishii and Minoru Kobayashi. ClearBoard: a seamless medium for shared drawing and conversation with eye contact. In *Proc. CHI '92*, pages 525–532, 1992.

[92] Hiroshi Ishii and Brygg Ullmer. Tangible Bits: Towards Seamless Interfaces between People, Bits and Atoms. In *Proc. UIST '97*, 1997.

[93] Tomáš Ižo and Jay Yagnik. Smart Thumbnails on YouTube, 2009.

[94] Ece Kamar, Severin Hacker, and Eric Horvitz. Combining Human and Machine Intelligence in Large-scale Crowdsourcing. In *Proc. AAMAS '12*, 2012.

[95] B Kanefsky, N.G. Barlow, and V.C. Gulick. Can Distributed Volunteers Accomplish Massive Data Analysis Tasks? In *Lunar and Planetary Institute Science Conference Abstracts*, volume 32 of *Lunar and Planetary Inst. Technical Report*, March 2001.

[96] Steven J Karau and Kipling D Williams. Social loafing: A meta-analytic review and theoretical integration. *Journal of Personality and Social Psychology*, 65(4):681–706, 1993.

[97] Michael Kearns, Siddharth Suri, and Nick Montfort. An experimental study of the coloring problem on human subject networks. *Science*, 313(5788):824–827, 2006.

[98] Melanie Kellar, Carolyn Watters, and Michael Shepherd. A field study characterizing Web-based information-seeking tasks. *JASIST*, 58(7):999–1018, May 2007.

[99] J F Kelley. An iterative design methodology for user-friendly natural language office information applications. *ACM Transactions on Information Systems (TOIS)*, 2(1):26–41, 1984.

[100] Do-kyum Kim, Marti Motoyama, Geoffrey M. Voelker, and Lawrence K. Saul. Topic modeling of freelance job postings to monitor web service abuse. In *Proc. Security and Artificial Intelligence*. ACM, 2011.

[101] Aniket Kittur. Crowdsourcing, collaboration and creativity. *XRDS: Crossroads, The ACM Magazine for Students*, 17(2):22–26, 2010.

[102] Aniket Kittur, Ed H. Chi, and Bongwon Suh. Crowdsourcing user studies with Mechanical Turk. In *Proc. CHI '08*, 2008.

[103] Aniket Kittur, Susheel Khamkar, Paul André, and Robert E. Kraut. CrowdWeaver: visually managing complex crowd work. In *Proc. CSCW '12*. ACM, 2012.

[104] Aniket Kittur, Boris Smus, Susheel Khamkar, and Robert E. Kraut. CrowdForge. In *Proc. UIST '11*, October 2011.

[105] Aniket Kittur, Bongwon Suh, Bryan A. Pendleton, and Ed H. Chi. He says, she says: conflict and coordination in Wikipedia. In *Proc. CHI '07*, 2007.

[106] Kevin Knight and Daniel Marcu. Summarization beyond sentence extraction:a probabilistic approach to sentence compression. *Artificial Intelligence*, 139(1), 2002.

[107] Aaron M Koblin. The sheep market. In *Proc. Creativity and Cognition '09*, 2009.

[108] Alex Kosorukoff. Human based genetic algorithm. In *Proc. Systems, Man, and Cybernetics*, volume 5. IEEE, 2001.

[109] Robert E. Kraut and Paul Resnick. *Building Successful Online Communities: Evidence-Based Social Design.* The MIT Press, 2012.

[110] Michel Krieger, Emily Margarete Stark, and Scott R. Klemmer. Coordinating tasks on the commons: designing for personal goals, expertise and serendipity. In *Proc. CHI '09*, 2009.

[111] Travis Kriplean, Ivan Beschastnikh, and David W McDonald. Articulations of wikiwork: uncovering valued work in wikipedia through barnstars. In *Proc. CSCW '08.* ACM, 2008.

[112] Karen Kukich. Techniques for automatically correcting words in text. *ACM Computing Surveys (CSUR)*, 24(4), 1992.

[113] Anand Kulkarni. The Complexity of Crowdsourcing: Theoretical Problmes in Human Computation. In *Proc. CHI '11 Workshop on Crowdsourcing and Human Computation*, 2011.

[114] Anand Kulkarni, Matthew Can, and Björn Hartmann. Collaboratively crowdsourcing workflows with Turkomatic. In *Proc. CSCW '12*, February 2012.

[115] Cliff A.C. Lampe, Nicole Ellison, and Charles Steinfield. A familiar face(book): profile elements as signals in an online social network. In *Proc. CHI '07*, 2007.

[116] Leslie Lamport. *LaTeX: A Document Preparation System*, volume 14. Addison-Wesley, 1994.

[117] Walter S. Lasecki, Kyle I. Murray, Samuel White, Robert C. Miller, and Jeffrey P. Bigham. Real-time crowd control of existing interfaces. In *Proc. UIST '11*. ACM Press, October 2011.

[118] Walter S Lasecki, Samuel C White, Kyle I Murray, and Jeffrey P Bigham. Crowd memory: Learning in the collective. *Proc. Collective Intelligence 2012*, 2012.

[119] Bibb Latane, Kipling Williams, and Stephen Harkins. Many hands make light the work: The causes and consequences of social loafing. *Journal of Personality and Social Psychology*, 37(6):822, 1979.

[120] Bibb Latane and John M Darley. *The Unresponsive Bystander: Why doesn't he help?* Appleton-Century Crofts, New York, 1970.

[121] Bibb Latane, Kipling Williams, and Stephen Harkins. Many hands make light the work: The causes and consequences of social loafing. *Journal of Personality and Social Psychology*, 37(6):822–832, 1979.

[122] Edith Law and Luis von Ahn. Human computation. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 5(3):1–121, 2011.

[123] Edith Law and Haoqi Zhang. Towards Large-Scale Collaborative Planning: Answering High-Level Search Queries Using Human Computation. In *Proc. AAAI '11*, 2011.

[124] John Le, Andy Edmonds, Vaughn Hester, and Lukas Biewald. Ensuring quality in crowdsourced search relevance evaluation: The effects of training question distribution. In *Proc. SIGIR 2010 Workshop on Crowdsourcing for Search Evaluation*, pages 21–26, 2010.

[125] Jane Li, Scott Huffman, and Akihito Tokuda. Good abandonment in mobile and PC internet search. In *Proc. SIGIR '09*, July 2009.

[126] Henry Lieberman. *Your wish is my command: Programming by example.* Morgan Kaufmann, 2001.

[127] Jimmy Lin. An exploration of the principles underlying redundancy-based factoid question answering. *ACM TOIS*, 25(2), April 2007.

[128] Greg Little, Lydia Chilton, Max Goldman, and Robert C Miller. Exploring iterative and parallel human computation processes. In *Proc. HCOMP '10*, 2010.

[129] Greg Little, Lydia Chilton, Max Goldman, and Robert C. Miller. TurKit: Human Computation Algorithms on Mechanical Turk. In *Proc. UIST '10*. ACM Press, 2010.

[130] Greg Little, Lydia B. Chilton, Max Goldman, and Robert C. Miller. TurKit: tools for iterative tasks on mechanical Turk. *Proc. HCOMP '09*, 2009.

[131] Greg Little, Tessa A. Lau, Allen Cypher, James Lin, Eben M. Haber, and Eser Kandogan. Koala: Capture, Share, Automate, Personalize Business Processes on the Web. *Proc. CHI '07*, 2007.

[132] Thomas W. Malone, Robert Laubacher, and Chrysanthos N. Dellarocas. Harnessing Crowds: Mapping the Genome of Collective Intelligence. *SSRN Electronic Journal*, 2009.

[133] Charles F Manski. Interpreting the predictions of prediction markets. Technical report, National Bureau of Economic Research, 2004.

[134] Andrew Mao, David C. Parkes, Ariel D. Procaccia, and Haoqi Zhang. Human Computation and Multiagent Systems: An Algorithmic Perspective. In *Proc. AAAI '11*, 2011.

[135] Daniel Marcu. *The Theory and Practice of Discourse Parsing and Summarization.* MIT Press, 2000.

[136] Adam Marcus, David R Karger, Samuel Madden, Robert C Miller, and Sewoong Oh. Counting with the Crowd. In *In Submission to VLDB*, 2012.

[137] Adam Marcus, Eugene Wu, David R Karger, Samuel Madden, and Robert C Miller. Human-powered sorts and joins. *Proceedings of the VLDB Endowment*, 5(1):13–24, 2011.

[138] Adam Marcus, Eugene Wu, David R Karger, Samuel R Madden, and Robert C Miller. Crowdsourced databases: Query processing with people. In *Proc. CIDR '11*. CIDR, 2011.

[139] P Markey. Bystander intervention in computer-mediated communication. *Computers in Human Behavior*, 16(2):183–188, March 2000.

[140] Winter Mason and Siddharth Suri. A Guide to Conducting Behavioral Research on Amazon's Mechanical Turk. *Social Science Research Network*, 1691163, 2010.

[141] Winter Mason and Duncan J Watts. Financial Incentives and the Performance of Crowds. In *Proc. HCOMP '09*. ACM Press, 2009.

[142] Ian McGraw, Chia-ying Lee, Lee Hetherington, Stephanie Seneff, and James R. Glass. Collecting voices from the cloud. In *Proc. LREC*, volume 100, 2010.

[143] Robert C Miller and Brad A Myers. Interactive simultaneous editing of multiple text regions. In *Proc. USENIX '01*, 2001.

[144] Vincent Miller. New media, networking and phatic culture. *Convergence: The International Journal of Research into New Media Technologies*, 14(4):387–400, 2008.

[145] Patrick Minder and Abraham Bernstein. CrowdLang—First Steps Towards Programmable Human Computers for General Computation. In *Proc. HCOMP '11*, 2011.

[146] Meredith Ringel Morris and Eric Horvitz. SearchTogether: an interface for collaborative web search. In *Proc. UIST '07*. ACM, 2007.

[147] Robert R Morris and Rosalind W Picard. Crowdsourcing Collective Emotional Intelligence. In *Proc. Collective Intelligence 2012*, volume abs/1204.3, 2012.

[148] E V Nalimov, C Wirth, G M C Haworth, and Others. KQQKQQ and the Kasparov-World Game. *ICGA Journal*, 22(4):195–212, 1999.

[149] Prayag Narula, Philipp Gutheim1, David Rolnitzky, Anand Kulkarni, and Bjoern Hartmann. MobileWorks: A Mobile Crowdsourcing Platform for Workers at the Bottom of the Pyramid. In *Proc. HCOMP '11*, 2011.

[150] Jakob Nielsen. *Usability engineering*. Morgan Kaufmann, 1993.

[151] Jakob Nielsen. Ten usability heuristics. 2005.

[152] Jon Noronha, Eric Hysen, Haoqi Zhang, and Krzysztof Z. Gajos. Platemate: crowdsourcing nutritional analysis from food photographs. In *Proc. UIST '11*. ACM, 2011.

[153] Ory Okolloh. Ushahidi, or 'testimony': Web 2.0 tools for crowdsourcing crisis information. *Participatory Learning and Action*, 59(1):65–70, 2009.

[154] Christopher Olston, Benjamin Reed, Utkarsh Srivastava, Ravi Kumar, and Andrew Tomkins. Pig latin: a not-so-foreign language for data processing. In *Proc. SIGMOD '08*. ACM, 2008.

[155] Katrina Panovich, Robert C Miller, and David R Karger. Tie strength in question & answer on social network sites. In *Proc. CSCW '12*. ACM, 2012.

[156] Aditya Parameswaran, Hector Garcia-Molina, Hyunjung Park, Neoklis Polyzotis, Aditya Ramesh, and Jennifer Widom. CrowdScreen: Algorithms for Filtering Data with Humans.

[157] Aditya Parameswaran, Hyunjung Park, Hector Garcia-Molina, Neoklis Polyzotis, and Jennifer Widom. Deco: Declarative Crowdsourcing. 2011.

[158] Manoj Parameswaran and Andrew B Whinston. Research issues in social computing. *Journal of the Association for Information Systems*, 8(6):336–350, 2007.

[159] Ioana Patringenaru and Tiffany Fox. *UC San Diego Team's Effort in DARPA's Shredder Challenge Derailed by Sabotage.* http://www.jacobsschool.ucsd.edu/news/news_releases/release.sfe?id=1150, 2012.

[160] Peter Pirolli. *Information foraging theory: adaptive interaction with information.* Oxford Press, 2007.

[161] Alex J Quinn and Benjamin B Bederson. Human computation: a survey and taxonomy of a growing field. In *Proc. CHI '11*. ACM, 2011.

[162] Al M. Rashid, Kimberly Ling, Regina D. Tassone, Paul Resnick, Robert Kraut, and John Riedl. Motivating participation by displaying the value of contribution. In *Proc. CHI '06*. ACM Press, 2006.

[163] Paul Resnick and Richard Zeckhauser. Trust among strangers in Internet transactions: Empirical analysis of eBay's reputation system. *Advances in Applied Microeconomics*, 11:127–157, 2002.

[164] J Rocchio. Relevance feedback in information retrieval. In Gerard Salton, editor, *The SMART retrieval system: experiments in automatic document processing*, pages 313–323. Prentice Hall, Englewood Cliffs, NJ, 1971.

[165] Mario Rodriguez and James Davis. CrowdSight: Rapidly Prototyping Intelligent Visual Processing Apps. In *Proc. HCOMP '11*, 2011.

[166] Yvonne Rogers, Helen Sharp, and Jenny Preece. Interaction Design: Beyond Human Computer Interaction. 2002.

[167] Joel Ross, Lilly Irani, M. Six Silberman, Andrew Zaldivar, and Bill Tomlinson. Who Are the Crowdworkers? Shifting Demographics in Amazon Mechanical Turk. In *alt.chi '10*. ACM Press, 2010.

[168] Brian C Russell, Antonio Torralba, Kevin P Murphy, and William T Freeman. LabelMe: a database and web-based tool for image annotation. *International journal of computer vision*, 77(1):157–173, 2008.

[169] Jeffrey M Rzeszotarski and Aniket Kittur. Instrumenting the crowd: using implicit behavioral measures to predict task performance. In *Proc. UIST '11*. ACM, 2011.

[170] Matthias Sala, Kurt Partridge, Linda Jacobson, and James Begole. An Exploration into Activity-Informed Physical Advertising Using PEST. In *Pervasive '07*, volume 4480 of *Lecture Notes in Computer Science*, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

[171] Matthew J Salganik and Karen E C Levy. Wiki surveys: Open and quantifiable social data collection. *Arxiv preprint arXiv:1202.0500*, 2012.

[172] Gerard Salton and Christopher Buckley. Term-weighting approaches in automatic text retrieval. *Information Processing and Management*, 24(5):513–523, 1988.

[173] Eric Schurman and Jake Brutlag. Performance related changes and their user impact. In *Velocity Web Performance and Operations Conference*, 2009.

[174] Victor S Sheng, Foster Provost, and Panagiotis G Ipeirotis. Get another label? improving data quality and data mining using multiple, noisy labelers. In *Proc. KDD '08*, pages 614—-622. ACM, 2008.

[175] M Six Silberman, Lilly Irani, and Joel Ross. Ethics and tactics of professional crowdwork. *XRDS*, 17(2):39–43, December 2010.

[176] M. Six Silberman, Joel Ross, Lilly Irani, and Bill Tomlinson. Sellers problems in human computation markets. In *Proc. HCOMP '10*. ACM, 2010.

[177] Herb A. Simon. Rational choice and the structure of the environment. *Psychological review*, 63(2):129, 1956.

[178] Ian Simon, Dan Morris, and Sumit Basu. MySong: automatic accompaniment generation for vocal melodies. In *Proc. CHI '08*, 2008.

[179] Push Singh, Thomas Lin, Erik T. Mueller, Grace Lim, Travell Perkins, and Wan Li Zhu. Open Mind Common Sense: Knowledge acquisition from the general public. *On the Move to Meaningful Internet Systems 2002: CoopIS, DOA, and ODBASE*, pages 1223–1237, 2002.

[180] Deborah A. Small, George Loewenstein, and Paul Slovic. Sympathy and callousness: The impact of deliberative thought on donations to identifiable and statistical victims. *Organizational Behavior and Human Decision Processes*, 102(2):143–153, 2007.

[181] Rion Snow, Brendan O'Connor, Daniel Jurafsky, and Andrew Y. Ng. Cheap and fastbut is it good?: evaluating non-expert annotations for natural language tasks. In *Proc. ACL '08*, 2008.

[182] Alexander Sorokin, Dmitry Berenson, Siddhartha Srinivasa, and Martial Hebert. People helping robots helping people: Crowdsourcing for grasping novel objects. In *Proc. IROS '10*, 2010.

[183] Alexander Sorokin and David Forsyth. Utility data annotation with Amazon Mechanical Turk. *Proc. CVPR '08*, 2008.

[184] Sofia Stamou and Efthimis N Efthimiadis. Queries without clicks: Successful or failed searches. In *Proc. SIGIR '09 Workshop on the Future of IR Evaluation*, 2009.

[185] Ruben Stranders, Sarvapali D. Ramchurn, Bing Shi, and Nicholas R. Jennings. CollabMap: Augmenting Maps Using the Wisdom of Crowds. In *Proc. HCOMP '11*, 2011.

[186] Fabian M Suchanek, Gjergji Kasneci, and Gerhard Weikum. YAGO : A Core of Semantic Knowledge Unifying Wikipedia and WordNet. In *Proc. WWW '07*, 2007.

[187] James Surowiecki. *The Wisdom of Crowds*. Random House, New York, 2005.

[188] Ivan E Sutherland. Sketch pad a man-machine graphical communication system. In *Proceedings of the SHARE design automation workshop*, pages 6–329. ACM, 1964.

[189] John C. Tang, Manuel Cebrian, Nicklaus A. Giacobe, Hyun-Woo Kim, Taemie Kim, and Douglas "Beaker" Wickert. Reflecting on the DARPA Red Balloon Challenge. *Communications of the ACM*, 54(4):78, April 2011.

[190] Jaime Teevan, Daniel J. Liebling, and Gayathri Ravichandran Geetha. Understanding and predicting personal navigation. In *Proc. WSDM '11*, February 2011.

[191] Michael Toomim, Travis Kriplean, Claus Pörtner, and James A. Landay. Utility of Human-Computer Interactions: Toward a Science of Preference Measurement. In *Proc. CHI 2011*, 2011.

[192] M M Tseng, R J Jiao, and C Wang. Design for mass personalization. *CIRP Annals-Manufacturing Technology*, 59(1):175–178, 2010.

[193] Kathleen Tuite, Noah Snavely, Dun-yu Hsiao, Nadine Tabing, and Zoran Popović. PhotoCity: Training experts at large-scale image acquisition through a competitive game. In *Proc. CHI '11*. ACM, 2011.

[194] Max Van Kleek, Michael Bernstein, M C Schraefel, and David R Karger. GUI–Phooey!: The Case for Text Input. In *Proc. UIST '07*, 2007.

[195] Simine Vazire and Samuel D. Gosling. e-Perceptions: Personality impressions based on personal websites. *Journal of Personality and Social Psychology*, 87(1):123–132, 2004.

[196] Fernanda B Viegas, Martin Wattenberg, Frank van Ham, Jesse Kriss, and Matt McKeon. ManyEyes: a site for visualization at internet scale. *IEEE Transactions on visualization and computer graphics*, 13(6):1121–1128, 2007.

[197] Jesse Vig, Shilad Sen, and John Riedl. Tagsplanations: explaining recommendations using tags. In *Proc. IUI '09*. ACM Press, 2009.

[198] Luis von Ahn. Games with a Purpose. *Computer*, 39(6):92–94, 2006.

[199] Luis von Ahn and Laura Dabbish. Labeling images with a computer game. In *CHI '04*, 2004.

[200] Luis von Ahn, Benjamin Maurer, Colin McMillen, David Abraham, and Manuel Blum. recaptcha: Human-based character recognition via web security measures. *Science*, 321(5895):1465–1468, 2008.

[201] Paul Wais, Shivaram Lingamneni, Duncan Cook, Jason Fennell, Benjamin Goldenberg, Daniel Lubarov, David Marin, and Hari Simons. Towards Building a High-Quality Workforce with Mechanical Turk. In *Proc. NIPS Workshop on Computational Social Science and the Wisdom of Crowds*, 2010.

[202] Jing Wang, Siamak Faridani, and Panagiotis G Ipeirotis. Estimating the Completion Time of Crowdsourced Tasks Using Survival Analysis Models. In *Proc. Crowdsourcing for Search and Data Mining '11*, 2011.

[203] Peter Welinder, Steve Branson, Serge Belongie, and Pietro Perona. The multidimensional wisdom of crowds. *Neural Information Processing Systems*, 6(7):1—-9, 2010.

[204] Ryen W. White, Mikhail Bilenko, and Silviu Cucerzan. Studying the use of popular destinations to enhance web search interaction. In *Proc. SIGIR '07*, 2007.

[205] Tingxin Yan, Vikas Kumar, and Deepak Ganesan. CrowdSearch: exploiting crowds for accurate real-time image search on mobile phones. In *Proc. MobiSys '10*. ACM, 2010.

[206] Lisa Yu and Jeffrey V Nickerson. Cooks or cobblers?: crowd creativity through combination. In *Proc. CHI '11*. ACM, 2011.

[207] Haoqi Zhang, Edith Law, Robert C Miller, Krzysztof Z Gajos, David C Parkes, and Eric Horvitz. Human Computation Tasks with Global Constraints. In *Proc. CHI '12*, 2012.

[208] Chen-Bo Zhong. The ethical dangers of deliberative decision making. *Administrative Science Quarterly*, 56(1):1–25, 2011.