Adaptive Crowd Algorithms for Open-Ended Problems

Lydia Beth Chilton

A dissertation
submitted in partial fulfillment of the
requirements for the degree of

Doctor of Philosophy

University of Washington

2016

Reading Committee:

James A. Landay, Chair

Daniel S. Weld, Chair

Michael S. Bernstein

Program Authorized to Offer Degree:
Computer Science & Engineering

University of Washington

Abstract

Adaptive Crowd Algorithms for Open-Ended Problems

Lydia Beth Chilton

Co-Chairs of the Supervisory Committee:
Professor James A. Landay
Computer Science and Engineering

Professor Daniel S. Weld
Computer Science and Engineering

Decomposing problems is fundamental to solving them for both people and computers. When it comes to problem solving, people and computers have complementary approaches. Computer algorithms can methodically solve large problems that require lots of state, organization, and memory, but they can only solve problems that are well-defined and have explicit steps. People are not quite as methodical, but can solve problems that are ill-defined and open-ended. This dissertation contributes concepts and techniques, embodied in software artifacts, to answer the following research question:

How we can combine the complementary skills of people and computers to solve open-ended problems systematically?

From the literature on human problem solving, design, and sensemaking, we know that the process people use to solve open-ended problems is not linear but iterative. People start with the concrete context of the situation to generate ideas, then they dynamically discover the parameters of the problem and adapt to them. To combine people's and computers' abilities, we must decompose the process of solving open-ended problems into explicit steps like an algorithm, but integrate human intelligence for the steps that computers cannot yet do.

My dissertation shows how to systematically solve open-ended problems with *adaptive crowd algorithms*. Adaptive crowd algorithms use crowdsourced microtasks to explore a solution space in incremental steps and test solutions until the goal is met. Because the problems are often large, workers are given only partial information about the problem and respond with proposals for partial solutions. Partial solutions can be tested against the goal and built upon by future microtasks to explore more of the solution space. To arrive at a cohesive output, *adaptive mechanisms* use partial solution to iterate towards the goal by exploring multiple ideas, testing ideas, adapting to feedback and nudging the output into a tested solution.

To demonstrate my thesis, I introduce three systems that systematically solve open-ended problems with adaptive crowd algorithms:

- Cascade crowdsources the open-ended problem of taxonomy creation.
- Frenzy coordinates a crowd of experts to meet the constraint of organizing accepted conference papers into thematic sessions.
- HumorTools decomposes the creative task for writing humorous news satire in the style of *The Onion*.

My evaluation shows that with adaptive crowd algorithms, we can solve open-ended problems too big for one person, too ill-defined to automate and that require creativity.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

I would like to thank my advisors, Dan Weld and James Landay (who I secretly refer to as "Danday") for giving me the freedom to follow my interests. Along the way, they exposed to me big ideas that I needed to solve this puzzle. I learned to appreciate design in Beijing at MSR Asia with James. I learned to frame problems in formal ways from Dan. That's just the tip of the iceberg.

I have been incredibly lucky to be supported by incredible friends and educators at a number of institutions: El Cerrito High School, MIT Economics Department, MIT EECS, University of Washington, Microsoft Research, Microsoft Research Asia, Google, and Stanford University. Through this long journey, so many people have gone out of their way to help me (in somewhat chronological order): Bob Fabini, Paul Joskow, Denny Ellerman, Randy Davis, David Karger, Daniel Jackson, Patrick Henry Winston, John Guttag, Rob Miller, Clayton Sims, Greg Little, Max Goldman, John Horton, James Landay, James Fogarty, Dan Weld, Gaetano Borriello, Kate Starbird, Shiri Azenkot, Nikki Dell, Daniel Epstein, Mike Toomim, Morgan Dixon, Tony Fader, Jesse Dodge, Katie O'Leary, Felicia Cordeiro, Chloe Kiddon, Jonathan Bragg, Kristi Gable, Katlyn Edwards, Janelle Van Hofwegen, Darren Edge, Sergio Paolantonio, Ed Chi, David Huynh, Eric Horvitz, Jaime Teevan, Shamsi Iqbal, Sumit Basu, Haoqi Zhang, Paul Andre, Steven Dow, Juho Kim, Michael Bernstein, Maneesh Agrawala, Stu Card, Jessica Cauchard, Bob West, Jane E, Evan Strasnick, Danae Metaxa, Sanjay Kairam, Kesler Tanner, Joy Kim, Niloufar Salehi, and Alex Tamkin.

I'd also like to thank my parents. My father checked my homework every night to make sure I was actually learning what I was meant to be learning. My mother is always supportive and understanding, and got me through hard times that no one else could. LL&P.

Chapter 1
INTRODUCTION

Algorithms are central to computation because an algorithm contains a procedure for solving a problem that is so precisely decomposed into well-defined steps that a computer can follow it. The steps are explicit and they flow together such that there are no mysterious leaps between them. As a consequence, algorithms can be repeated, analyzed, debugged, optimized, and distributed across machines. Algorithms excel at solving problems in the ways people, in contrast, often fail. Algorithms can do the same operations for days without error, they can process large amounts of data without being overwhelmed, they can switch between tasks seamlessly, and they can be written to be distributed across machines with minimal coordination costs.

But algorithms can only solve problems that are well-defined. The problem and relevant inputs are stated, the goal and outputs are clear, the process must be perfectly decomposed into explicit steps. This works for numerical calculations, networking protocols, searching data structures, and transforming data. But we know that many problems do not meet these criteria. These problems are ill-defined and open-ended.

Problems are open-ended when the goal is not explicitly defined and thus there are multiple acceptable answers. The goal may be vague and thus the outputs are subjective – they are not correct or incorrect but good or bad. For example, writing a joke and organizing personal data are both open-ended problems. The goal of a joke is not explicitly defined. There is no definition for what is funny. There are many possible jokes, some better than others. It is also subjective – the same things are not equally funny to everybody. The goal state of organization is also not explicitly defined. There is no standard for "good"

organization. There are many acceptable organizations. The output is dependent on the details of the input. If the photos are abstract and colorful it might be best to organize them by color, but if not, it may be better to organize them by theme. For open-ended problems, the vagueness of the problem statement and the subjective nature of the solution means that there is no well-defined problem space for a traditional algorithm to search in or operate on to find a solution. Framing the problem is part of the problem. Thus, open-ended problems are beyond the scope of traditional algorithms. However, our everyday experience shows that people clearly have the skills to solve open-ended problems.

Instead of operating in a well-defined problem space, the strategy people use to solve open-ended problems involves dynamically discovering the parameters of the problem and adapting to them. A large class of open-ended problem are design problems, and to address them, designers use iterative models such as Human-Centered Design [**?**] to progressively understand and solve problems. This design process involves:

- observation
- generating ideas
- testing ideas
- adapting the next step based on the feedback

Because the problem is usually ill-defined, the designer starts by observing the situation to define a problem. This cannot be done abstractly. Designers must observe the context, environment, and details of a situation. Based on partial information and contextual clues, they can generate ideas for the problem or how to solve them. Even after defining a problem, there are many unknowns. Through testing ideas, new parameters, constraints, and complications arise that cannot have been foreseen, and designers must find ways to adapt to the new information. Thus, linear methods like the waterfall method where steps are planned from the beginning and then executed will not work. Designers must constantly discover the problem through observation of context and testing and adapt their approach.

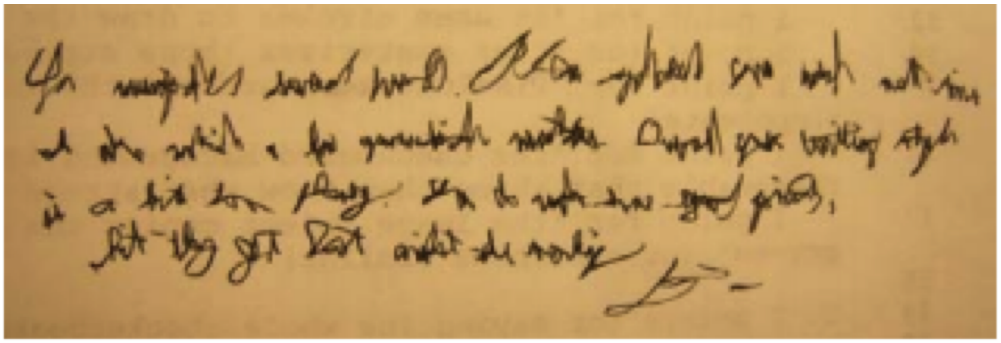
Even without full information, people can use contextual clues, and apply heuristics from their world knowledge to make reasonable ideas to test.

Although people can solve open-ended problems, they often get stuck. As problems get large, it becomes hard to manage the state, information, and context necessary to solve the problem. Also, since people often do not know or articulate the steps of their process, it can be hard to bring in new perspectives or get them unstuck. The question this dissertation addresses is:

How we can combine the complementary skills of people and computers to solve open-ended problems systematically?

To do this, we must decompose the process of solving open-ended problems into explicit steps like an algorithm, but integrate human intelligence for the steps that computers cannot yet do.

Crowdsourcing projects like The ESP Game [**?**] and Galaxy Zoo [**?**] use human labor to complete specific tasks that computers cannot yet solve such as image labeling and classification. Microtasks enable millions of people to participate by doing easy tasks with a short time commitment, which is essential to crowdsourcing a large dataset. Early crowdsourcing research developed crowd algorithms [**?**] – ways to combine the results of several microtasks to produce more complex artifacts. The first crowd algorithm, Improve-and-Vote, used the crowd to collectively decipher blurry text that no one worker could decipher by having workers build on the contextual clues contributed by other workers [**?**]. (See Figure **??**.) Another early system shortened paragraphs of text by having multiple workers find, fix and verify areas that could be shortened [**?**]. Many early crowd algorithms decomposed problems into linear workflows. Regardless of the details of the input, the same workflow was used to edit text [**?**], extract instructions from videos [**?**], or count the calories in food [**?**]. These workflows were useful for doing some complex tasks that computers could not do. But linear approaches only work on problems that are well-defined: the problem space was clearly pa-



Iteration 1:

You (?) (?) (?) (work). (?) (?) (?) work (not) time. I (?) (?) a few grammatical mistakes. Overall your writing style is a bit too (phoney). You do (?) have good (points), but they got lost amidst the (writing). (signature)

Iteration 6:

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)

Figure 1.1: The crowd could decipher messy handwriting by having each worker build on the contextual clues of other workers.

rameterized and the steps can be executed regardless of the specific details of the problem. For open-ended problems, we need iterative workflows that dynamically discover the problem and adapt to it.

1.1 Thesis and Contributions

My dissertation shows how to systematically solve open-ended problems with *adaptive crowd algorithms*. Adaptive crowd algorithms use crowdsourced microtasks to explore a solution space in incremental steps and test solutions until the goal is met. Because the problems are often large, workers are given only partial information about the problem and respond with proposals for partial solutions. Partial solutions can be tested against the goal and built upon by future microtasks to explore more of the solution space. To arrive at a cohesive output, *adaptive mechanisms* use partial solution to iterate towards the goal by exploring multiple ideas, testing ideas, adapting to feedback and nudging the output into a tested solution.

To demonstrate my thesis, I introduce three systems that systematically solve open-ended problems with adaptive crowd algorithms:

- Cascade crowdsources the open-ended problem of taxonomy creation.
- Frenzy coordinates a crowd of experts to meet the constraint of organizing accepted conference papers into thematic sessions.
- HumorTools decomposes the creative task for writing humorous news satire in the style of *The Onion*.

The algorithms embodied in these systems use microtasks that present partial information about the problem, such as a single item in the dataset, and ask the crowd to contribute heuristics and contextual clues through microtasks such as voting, labeling, categorizing, evaluating, and brainstorming associations.

To combine the output of the microtasks into a cohesive solution, the systems use three different adaptive mechanisms:

- recursive generate-and-test
- flare-and-focus
- searching a conceptual space with backtracking until a constraint can be satisfied

Each of these adaptive mechanisms presents partial information to the crowd, gathers proposed partial solutions and tests the partial solutions to figure out which ones are appropriate and can adapt to new information. With adaptive crowd algorithms, we can solve open-ended problems too big for one person, too ill-defined to automate and that require creativity.

The contributions of the research described in this dissertation include the following:

Concepts and Techniques

- Adaptive Crowd Algorithms, a new approach to crowdsourcing that uses adaptive exploration rather than static workflows.
- Three microtask primitives, Generate, SelectBest, Categorize, that can be reused in other crowd algorithms.
- The recursive generate-and-test mechanism that achieves the global-from-local property that each worker sees only a local view of the data, but the algorithm produces a globally cohesive output.
- Using flare-and-focus as a adaptive mechanism to coordinate experts at solving an ill-defined data organization problem.
- Framing creativity as a constraint satisfaction problem that is met by searching a conceptual space of associations with backtracking to adapt to dead-ends.

Artifacts

- Cascade, an algorithm for crowdsourced taxonomy creation where each worker sees only a local view of the data, but the algorithm creates a global view of the data. This algorithm is highly parallelizable and can be run equally well on text or image data.

- Frenzy, a platform for a community of experts to participate in the parallel crowdsourcing of their own data organization.
- HumorTools, a web interface that decomposes the creative task of writing humor to help novices write humorous news satire.

Experimental Results

- A study comparing Cascade's taxonomies to those of expert information architects showing Cascade achieves 80-90% of expert quality.
- A deployment of Frenzy at the CSCW 2014 program committee meeting showed session-making, which usually takes 6 to 8 hours, was completed in only 88 minutes using Frenzy.
- A study of HumorTools with 20 participants showing that 75% of them could write jokes using the decomposed HumorTools process and 25% of the resulting jokes were broadly funny. In a direct comparison with *The Onion*, three of the nine best HumorTools headlines were voted funnier than *The Onion*.

1.2 Dissertation Roadmap

This section presents a brief overview of the structure of this dissertation by chapters.

1.2.1 Related Work

Problem solving has been a grand challenge in computer science since the 1950's, when Herb Simon and Allen Newell sought to understand human problem solving and decompose it into processes that computers could perform [**?**, **?**]. Today, problem solving still requires human intelligence. The fields of design and sensemaking both have processes people use to solve open-ended problems. We review the details of these methods and show how they both rely on adaptive and iterative processes rather than rigid, linear workflows.

In the Related Work chapter, we define microtask crowdsourcing, and give a history of crowd algorithms as they build towards increasing complexity. Lastly, we review current

approaches to the types of open-ended problems addressed in this dissertation: data organization and creativity. Our review includes both fully manual approaches and fully automated methods such as LDA for clustering [**?**] and computational humor for creativity [**?**, **?**, **?**, **?**].

1.2.2 Cascade: Problems Too Big for One Person

Taxonomies organize independent pieces of information into meaningful semantic structure to aid many types of information tasks such as information retrieval, browsing, and sense-making. The inputs to a taxonomy are well-defined: the items that need to be organized. However, the resulting taxonomy is open-ended and subjective. Even expert information architects do not agree on a single best taxonomy for a data set. Creating taxonomies is a large task for one person. There might be a large number of items to consider and there can be complex interdependencies between them that make it challenging to remember all of them. Cascade is an adaptive crowd algorithm that decomposes the process of creating a taxonomy into microtasks and distributes the pieces to the crowd [**?**].

The challenge in Cascade is to output a cohesive global organization of the data when each worker only sees a local subset of the data. This property of crowd algorithms is called *global-from-local*. To achieve this, Cascade uses three primitive microtasks: *Generate*, *SelectBest*, and *Categorize*. The adaptive mechanism that Cascade uses is *Recursive Generate-and-Test*. A worker is given a single data item and is asked to use their judgment to generate a potential category for that item. Multiple potential categories are generated for each data item from different workers, and the next microtask is to select the best potential category for a single data item. This step filters spam and vague categories early in the process. The categories from this SelectBest stage are generated based on a single data item and are not guaranteed to fit other items in the data set. The next step is for workers to vote on the categorization of every data item into every potential category. This generates a matrix of relationships between items and categories from which Cascade infers a taxonomy from overlapping members of items in categories. At this point, Cascade has generated and tested categories from the crowd to produce a small taxonomy. However, the workers who

Figure 1.2: Cascade Crowdsourced Taxonomy. The left represents the input: 100 nature photos. The right side is the resulting taxonomy with call outs showing examples of three categories: tiger, worker, and historical landmarks.

generated suggestions only had partial information about the data set, thus it is likely that the taxonomy will not fit all the data items. Next, Cascade identifies the data items that do not yet fit in the taxonomy, and recursively re-runs Cascade on those items to generate-and-test new categories that they will fit in.

Recursive generate-and-test is an adaptive strategy that relies on only partial information. It generates guesses locally and tests them globally. The recursive running of generate-and-test adaptively responds to updated information on which items do not yet fit in the taxonomy and need to be seen again.

Figure **??** is an example of the taxonomy produced by Cascade for 100 nature photos. In an evaluation on three datasets, Cascade taxonomies achieved 80-90% accuracy compared to four expert information architects who created taxonomies from the same data.

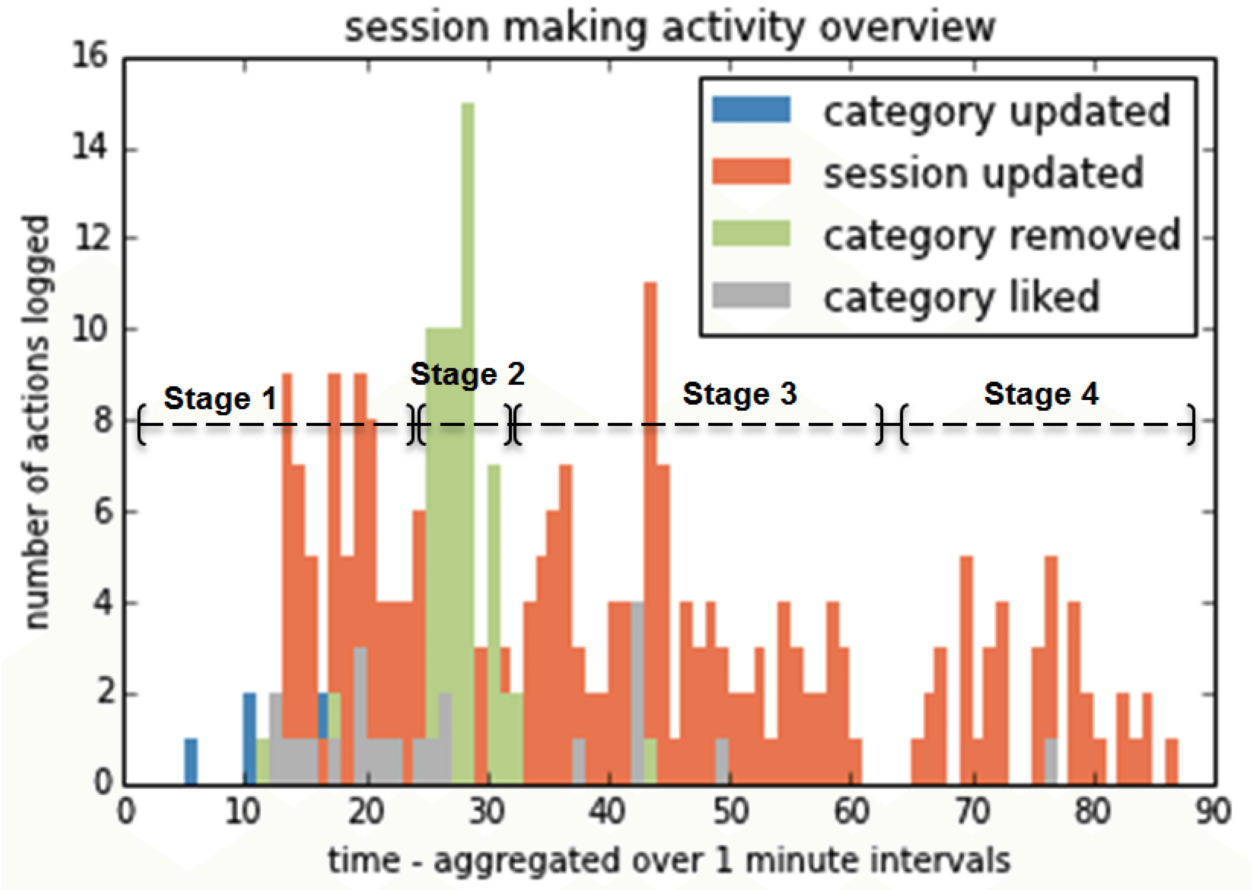


Figure 1.3: Activity during Frenzy session-making. Multiple interleaved task types over time show the dynamic nature of the Frenzy algorithm.

1.2.3 Frenzy: Problems Too Ill-Defined to Automate

Frenzy is a tool that helps experts collaboratively complete information organization tasks [**?**]. Frenzy was used by the CSCW 2014 and CHI 2014 conference program committee to group accepted papers into thematic conference sessions for these large multi-track conferences. Session-making is a more constrained information organization task than taxonomy creation. In taxonomies, categories can be large or small and can overlap in their members. But for conference session-making, for example, each session must have exactly four papers and each paper must be in exactly one session.

The inputs to Frenzy are well-defined: accepted conference papers. But the resulting session organization is open-ended and subjective. Moreover, the problem is ill-defined: there are constraints on the sessions that aren't known before hand. They are discovered dynamically by experts and the output must adjust accordingly. For example, at CSCW 2014, four papers were initially fit together in a session called "Collaborative Crowdsourcing." Thematically this would be a good session, however, the papers all had the same author, which made the session less than ideal. Once the experts saw this, they decided it was inappropriate to have this session. Because of dynamically discovered constraints like this one, session-making is too ill-defined to automate with a solution like Cascade. Instead, Frenzy uses *flare-and-focus* as an adaptive mechanism to help re-adjust categories once a new constraint is discovered.

Frenzy uses the same microtasks used in Cascade, but the presentation and agency of the workers is different. Frenzy is a social platform for trusted experts that looks and feels like Twitter. It presents an open stream of papers that experts can search and contribute to the resulting organization of by adding metadata to the papers they choose to examine. The metadata experts can add is equivalent to the microtasks used in Cascade: Generate, SelectBest and Categorize. Frenzy users contribute this data by adding labels (Generate), up-voting labels they like (SelectBest) and categorizing papers into labels and sessions (Categorize).

To guide the experts' microtasks into a cohesive output, Frenzy introduces the adaptive mechanism of flare-and-focus, a pattern adapted from the design literature, also known as diverge-converge [**?**]. Flare-and-focus scaffolds the work into two stages, each with a different goal: a flare stage, where the goal is to ideate many possible categories for papers, and a focus stage, where the goal is to meet the hard constraints of session-making.

In the flare stage, experts generate categories for papers by meeting a very loosely constrained goal: that every paper needs at least one category and every category needs at least two papers (no singleton categories). Unlike sessions, categories can be small or large with as few as two or as many as seventy papers in them and papers can be in multiple categories.

In the focus stage, the experts start with the categories from the flare stage and work towards meeting the tighter constraints of session-making: sessions must have exactly four papers, and each paper must be in exactly one session. Flare-and-focus allows for adaptation because when a session must be broken up due to a newly discovered constraint, experts can look back at the categories and metadata from the looser constraints to guide them forward.

Frenzy was deployed at the CSCW 2014 program committee meeting to create the sessions used at the conference. The session-making process usually takes 6-8 hours, but using Frenzy it took only 88 minutes. This speed up is due to the parallelization of effort, enabled by the lightweight contribution methods and the flare-and-focus mechanism. Figure **??** shows activity during the Frenzy session-making period at the CSCW 2014 program committee meeting. High levels of activity, particularly during the first three stages attest to how well Frenzy supported parallel contributions from the expert crowd.

1.2.4 HumorTools: Problems That Require Creativity

HumorTools is a system that decomposes the creative process of writing jokes into simple microtasks. Writing jokes is just one example of a problem that requires creativity to solve. Like other creative tasks, jokes are novel and difficult to produce. Creative tasks, such as writing humor, are open-ended. There are many possible jokes for a given inspiration and the quality of the resulting joke is subjective, although some jokes are clearly better or worse than others.

Creativity is thought to be hard to decompose. HumorTools frames the difficult and creative aspect of joke writing as a constraint satisfaction problem. Advice from linguists [**?**, **?**], comedians [**?**, **?**, **?**, **?**, **?**, **?**] and philosophers [**?**, **?**] suggest that a large class of jokes are funny because they satisfy the constraint of violating an expectation. To help meet this constraint, HumorTools provides users with microtasks that help search a conceptual space of associations. Users can search this space to find a way to violate an expectation. Not all associations will lead to an expectation violation, so users must adaptively backtrack when an association leads to a dead end and pick a new association to explore.

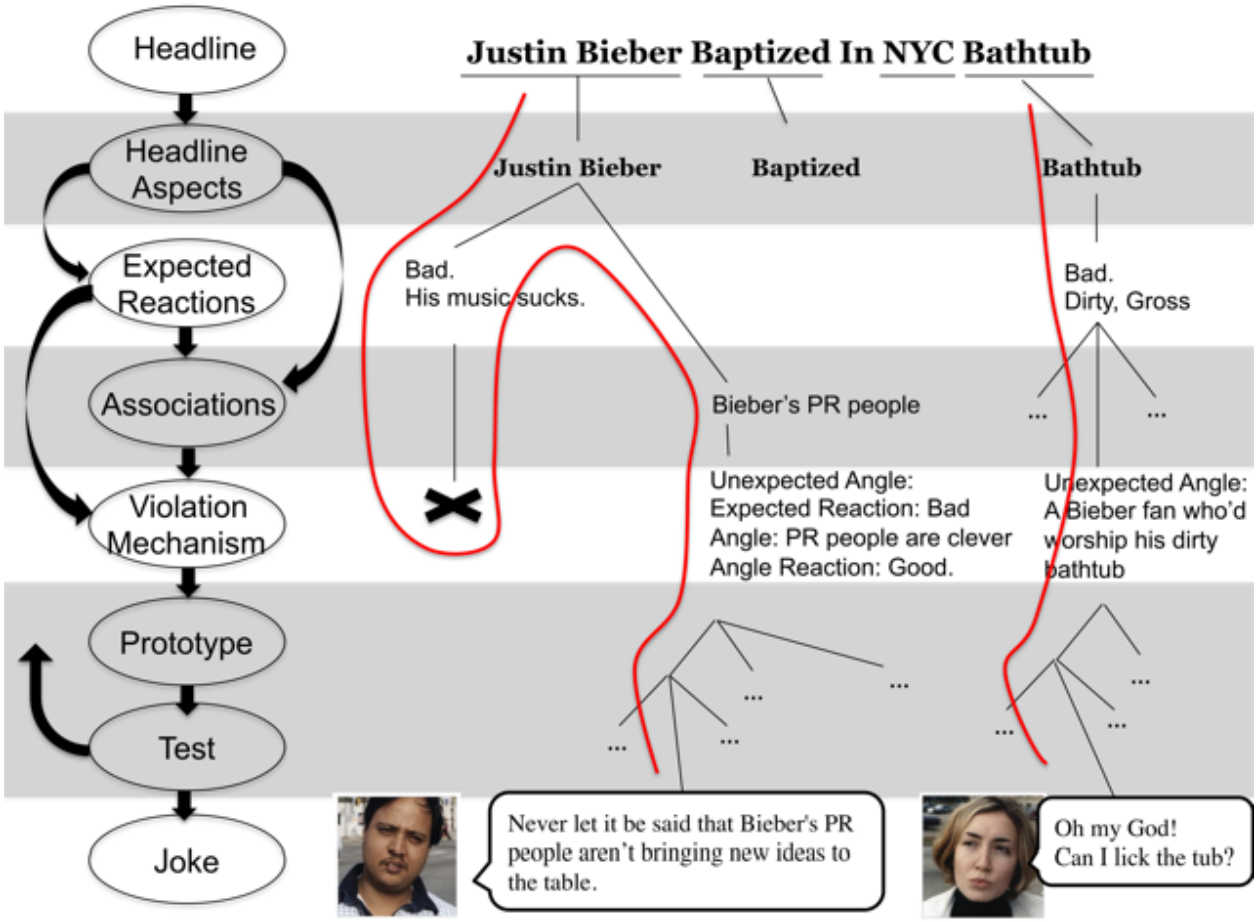


Figure 1.4: HumorTools uses microtasks in the iterative design process to help people write news satire. On the left are the stages of the iterative design process with arrows showing typical transitions between states. On the right is an illustration of the process of generating two jokes: an exploration of a conceptual space in a tree-like search structure. The red lines indicate the flow of activity, with the red line on the left indicating backtracking to find a humorous solution that satisfied the humor constraints.

The input to HumorTools is a real news headline. And the output is a joke in the style of *The Onion's* American Voices [**?**] jokes. For example:

> Real news headline: *"People Bending iPhones at Apple Stores"*
>
> *The Onion's* joke: *"I can't believe people would just walk into an Apple Store and start breaking things like it's a Best Buy."*

Although the headline is a well-defined input, the background of the headline, the assumptions the reader makes, and the associations with aspects of the headline are not explicit inputs and must be discovered and adapted to.

HumorTools introduces twenty microtasks users can apply to the headline or aspects of the headline to brainstorm associations. These include brainstorming alternatives (Best Buy is an alternative to Apple Stores as a consumer electronics retailer) and generating insults (Best Buy is low-end and disorganized). The goal of associations is to violate an expectation. This goal is vague, so HumorTools presents three concrete mechanisms for violating expectations. One of them is *Bait-and-Switch*: the joke first grounds the reader in their expectation (*"I can't believe people would just walk into an Apple Store and start breaking things?"*), but then switches to an unexpected association such as an insult (*"...like it's a Best Buy."*)

The adaptive mechanism in HumorTools is to use brainstorming to generate many partial solutions to satisfying the expectation violation constraint, and then to backtrack if the partial solution does not satisfy a constraint. (See Figure **??**) HumorTools was evaluated with 20 participants and demonstrated that 75% of them could write jokes using the decomposed HumorTools process and 25% of the resulting jokes were broadly funny. In a direct comparison with *The Onion*, three of the nine best HumorTools headlines were voted funnier than those from *The Onion*.

1.2.5 Future Work

We believe that in the future adaptive crowd algorithms can solve any problem where the solution space is known to contain design patterns. This includes software engineering, urban planning, graphic design, and screenwriting, to name a few. We believe there are many more problem domains, such as humor, that are not traditionally thought to have design patterns, but where design patterns can be found. This includes writing opinion pieces in for a newspaper, constructing a legal case, journalism, marketing, and scientific investigation. By using sensemaking tools like Cascade and Frenzy we can use the crowd to identify design patterns in new domains that we can teach the crowd to apply. For system and algorithms, adaptive crowd algorithms are a new way of decomposing problems, that uses people's conceptual abilities systematically. For crowdsourcing, adaptive crowd algorithms are a new way to combine people's ideas into a cohesive solution using search and design patterns.

Although we believe that many creative problems can be solved with adaptive crowd algorithms, it is possible that some cannot. Adaptive crowd algorithms rely on the workers in the crowd to have shared terms and knowledge of design patterns. However, some exceptionally novel solutions break the mold by not using traditional design patterns. Some solutions invent new techniques. For example, James Joyce's Ulysses [**?**] abandoned traditional prose structure for a new stream of consciousness style. Similarly, to prove the Halting Problem is undecidable, Alan Turing invented a new mechanism – Turing Machines [**?**]. Problems that require inventing new mechanisms are outside the scope of adaptive crowd algorithms.

HumorTools is the first step towards the ultimate goal of systematically generating creative artifacts. In the future, the HumorTools approach can be extended to other forms of humor and to new domains of creativity. The immediate next steps are to build on the existing HumorTools platform by crowdsourcing the process, and increasing the quality of the jokes. In the near term, we can begin to automate some or all of the microtasks in the process. After that, the HumorTools can be extended to other styles of humor such as *The*

New Yorker's caption contest and the satirical headlines that *The Onion* is traditionally known for.

Further in the future, we can generalize the HumorTools approach to other domains. We propose two domains: writing persuasive argumentation and conducting empirical economics research. Additionally, to enable people to decompose other creative domains, we need tools to support the decomposition process. This process is similar to sensemaking, and the tool we used to support the humor decomposition/sensemaking process was Frenzy [**?**]. We can build on the insights of Cascade [**?**] and Frenzy [**?**] for crowdsourcing data organization to design tools that support distributed sensemaking. This will enable us to crowdsource not only the generation of creative artifacts in chosen domains, but also the process for decomposing new domains of creative artifacts.

Chapter 2
RELATED WORK

This dissertation proposes novel algorithms for solving open-ended problems. To motivate this problem and put it into context, this chapter first presents background on methods for solving open-ended problems. Problem solving has been a grand challenge in computer science since the 1950's when Herb Simon and Allen Newell sought to understand human problem solving and decompose it into processes that computers could perform [**?**, **?**]. Today, problem solving still requires human intelligence. The fields of design and sensemaking both have processes people use to solve open-ended problems. We review the details of these methods and show how they both rely on adaptive and iterative processes rather than rigid, linear workflows. We introduce design patterns as a technique that experts use to solve open-ended problems by reusing previously discovered solutions to common problems.

The techniques proposed by this dissertation build on the literature from crowdsourcing. We define microtask crowdsourcing, and give a history of crowd algorithms as they build towards increasing complexity. We discuss lessons that crowdsourcing can draw from groupware, particularly how tacit knowledge is needed to solve problems within a community of experts.

Lastly, we review current approaches to the types of open-ended problems addressed in this dissertation: data organization and creativity. Our review includes both fully manual approaches and fully automated methods such as LDA for clustering and computational humor for creativity.

2.1 Background

The systems in this dissertation are built on the ideas of problem-solving, design, sensemaking and design patterns. In this section we give an introduction to each of these historical topics.

2.1.1 Problem-Solving

Problem-solving is a long-standing goal in computer science. In 1958, Allen Newell, J.C. Shaw, and Herbert Simon published "Elements of a Theory of Human Problem Solving" [**?**]. They argued that human problem-solving behavior, which is often considered mysterious and unexplained, could be decomposed into information processing steps that a computer program would eventually be able to complete. In their information processing paradigm, they modeled problem-solving as a *heuristic-based search* that involved:

1. "searching for possible solutions"
2. "generating possible solutions out of other elements"
3. "evaluating partial solutions and cues"

To demonstrate that problem solving could be achieved through search, they presented their program, Logic Theorist, for deriving mathematical proofs. Logic Theorist successfully proved 38 of the first 52 theorems in Chapter 2 of Russell and Whitehead's *Principia Mathematica* [**?**]. However, Logic Theorist worked on well-defined problems for which there was a known representation (logic). Further work would be needed to extend the heuristic search approach to other problems, but for this and other seminal work Simon and Newell won the 1975 Turing Award [**?**, **?**]

In 1971, Simon and Newell published an article reflecting on the state of the theory of human problem solving [**?**] that emphasized the important role of adaptivity in problem-solving. There are very few general mechanisms needed to do problem-solving. But the mechanisms can solve many problems because they can adapt to the task environment. For example, if the process to solve a problem is search, then the hard constraints of the task

environment are important to shaping the behavior of the problem-solving system. The task environment determines the problem space. Problem spaces are all different – they differ not only in size, but also in their structure. Some problem spaces have little structure, and require brute force search, whereas others have redundancy that can be used to reduce the search space. They describe the process used to solve problems in terms of adapting to the task environment. Problem-solving requires: "selective search in a problem space that incorporates some of the structural information of the task environment [**?**]." One challenge this poses is how to generate a problem space and what to do if a problem space is ill-defined and open-ended. This is a challenge addressed by design.

2.1.2 Design

Design is a framework for solving ill-defined and open-ended problems. Don Norman describes that an integral part of the design process is defining the problem. Thus, by definition design solves ill-defined problems [**?**]. The design process starts by understanding what the problem is. It does this not by working from an abstract description of the problem but by understanding the context and the environment of the problem. Design problems often have requirements that are vague and ill-defined, such as "the product needs to sell", indicate that the solution is open-ended. Buchanan speaks to the fact that design can also address ill-defined, open-ended, or "wicked" problems where the solutions are not true or false, but good or bad [**?**]. This makes the solutions inherently subjective. This is consistent with Simon's notion that for ill-defined problems it is not possible to enumerate all the possible options and compute a global optimum. Instead, solutions should *satisfice* – they are not optimal, but they are good enough. Particularly in problems that are over-constrained, there may not be a perfect solution. The evaluation is a relative judgment of fit, not a global optimum [**?**].

The design process is iterative rather than linear. Linear methods require rigid specifications from the beginning, but iterative methods allow the specification to evolve with the solution. Although linear processes have advantages of being simpler, their weakness

is that they are not adaptive. Norman states: "[In a linear model], progress goes in one direction, once decisions have been made, it is difficult or impossible to go back [**?**, p.234]." The iterative model has four stages:

- Observation
- Idea Generation
- Prototyping
- Testing

Designers use these stages iteratively by exploring the solution space before converging on a solution and backtracking when necessary. The iterative application of these stages helps the process adapt to the specific context of the problem, unforeseeable dead ends, and newly discovered constraints.

The iterative design process uses several mechanisms for adapting to the specific problem.

Observation

Observation is "the initial research to understand the nature of the problem itself [**?**, p.222]." By watching people in their natural environment (e.g., homes, schools, offices, and commuting) designers can better understand and characterize the problems people encounter and determine what problem to solve. By establishing the context of the problem concretely, the ideas and solutions can be molded or adapted to the details of the environment.

Idea generation

Instead of jumping to a solution, the design process favors generating many ideas before selecting a solution. Similar to brainstorming, idea generation should be unconstrained, because"[e]ven crazy ideas (wrong) can contain insights that can later be extracted and put to good use in the final idea selection' [**?**, p.226]." Beyond just enumerating potential solutions, ideation explores the problem space. Allowing one idea to build off aspects of another idea enables designers to discover not just single ideas but areas of opportunity to explore more

deeply. By questioning assumed constraints, ideation allows discovery of the true shape of the space. Exploring the problem-space rather than assuming a solution helps adapt the solution to the specific problem space.

Diverge-Converge

Effective design needs to satisfy a large number of constraints. Often products have multiple conflicting constraints [**?**, p.240]. Diverge-Converge [**?**, **?**], also known as flare-and-focus, frees designers from unnecessary constraints to the problem space and the solution space. In the diverge stage, ideation is important in brainstorming many options. In the converge stage, the ideas are used to converge on a definition of a problem or the delivery of a concrete solution. This allows designers to adapt both their ideas to the specific problem space and their solutions to the specific solution space.

Backtracking

Through iterative application of observation, ideation, prototyping, and testing, the design process continually refines ideas and encourages backtracking when tests fail. Iteration implies that the designers are running a generate-and-test loop where they learn from experience, and adapt as more problem context is discovered. Used together, these techniques can support exploration under the uncertainty of solving ill-defined and open-ended problems by discovering and adapting when a dead end is encountered.

Design researchers debate how to frame the design process. There are two major competing framings: the Rational Model derived from Simon and Newell [**?**] and the Reflection in Action model developed by Schon [**?**]. The Rational Model describes the role of the designer as a disembodied information processor. However, in the Reflection in Action Model, the designer is a part of the design process that cannot be factored out. The designer is capable of empathy and he embeds himself in the environment and is capable of embodied cognition [**?**]. He is reflective of his own process, he picks a way to frame the problem, then

he takes an action and reflects on it again [**?**]. Although the approaches disagree on whether a person is a necessary part of the process, they both agree that the process is iterative and adapt to new information. The limits of the information processing model are not yet known. There may be some inherent need for human capabilities such as embodied cognition, and there may not be.

2.1.3 Sensemaking

Sensemaking is a framework for making sense out of a large amount of information. Like design, sensemaking is an open-ended task. The goal is vaguely described as using the data to "build a case" or "tell a story" [**?**] and there is no single right answer. The input is typically a large amount of unstructured data, and "sensemaking is the process of searching for a representation and encoding data in that representation to answer task-specific questions [**?**]." For example, Russell et al. present a case study of doing sensemaking of the technical manuals for 21 laser printers and scanners to "design a new generic training course on laser printing for Xerox technicians [**?**]." There was clearly redundancy and structural similarity between the 21 devices, and by finding a unifying structure and terminology they could decrease training time. The sensemaking process took six months. It involved a lead group of analysts in addition to subject matter experts. It resulted in a representation of printers in terms of their individual components that are shared across the 21 printers and scanners.

Sensemaking is an iterative process. It involves four basic types of tasks titled Information, Schema, Insight and Product:

- **Information**: gathering information relevant to the goal
- **Schema**: "re-representation of the information in a schema that aids analysis."
- **Insight**: "the development of insight through the manipulation of this representation"
- **Product**: "the creation of some knowledge product or direct action based on the insight"

These tasks are part of an iterative process called a *Learning Loop*. The Learning Loop

is a process that describes how analysts iteratively search for representations of the information, and applies those representations across all the data looking for *residue* – information that does not fit the current representation. Residue is analyzed and used to improve the representation to better fit all the data [**?**]. (See Figure **??**).

Pirolli and Card [**?**] describe sensemaking as an opportunistic mix of bottom-up and top-down approaches. In the bottom-up processes, analysts search for representations by looking at the data. In the top-down processes, analysts instantiate the representations across all data. The opportunistic mix is what makes the process iterative and adaptive. Linear approaches are purely top-down: representations are decided from the outset, then simply applied to the data. If a linear, top-down approach worked, then sensemaking would be trivial [**?**]. The designer would merely define schemas, then instantiate them. But in practice, schemas must be revised when they do not generalize as planned or as new task requirements are discovered, which calls for an iterative approach.

The sensemaking process shares some aspects of adaptivity with the design process. "Information gathering" in sensemaking is similar to "Observation" in design. Both processes start by establishing the concrete context or environment that will inform future tasks. Generating ideas and testing them is prevalent in both processes. In sensemaking, the analyst ideates schemas and tests them on other data. In design, designers use the diverge-converge process to iteratively generate and test design ideas. Iteration is used in both processes. In sensemaking, learning loops are iterative in that they test schemas against data and must adapt the schema to the residue. In design, iteration is used for continual refining and enhancement of solutions to satisfy many different constraints.

One difference between sensemaking and the design process is that design does not have an explicit notion of what insights are. In sensemaking, insights come from finding a schema that fits the data. The notion of schema-as-insight is specific to information processing problems, but it is based on the general notion that during problem-solving, representation matters. In *The Sciences of the Artificial*, Simon claims that: "Solving a problem simply means representing it so as to make the solution transparent [**?**]." One theory for why

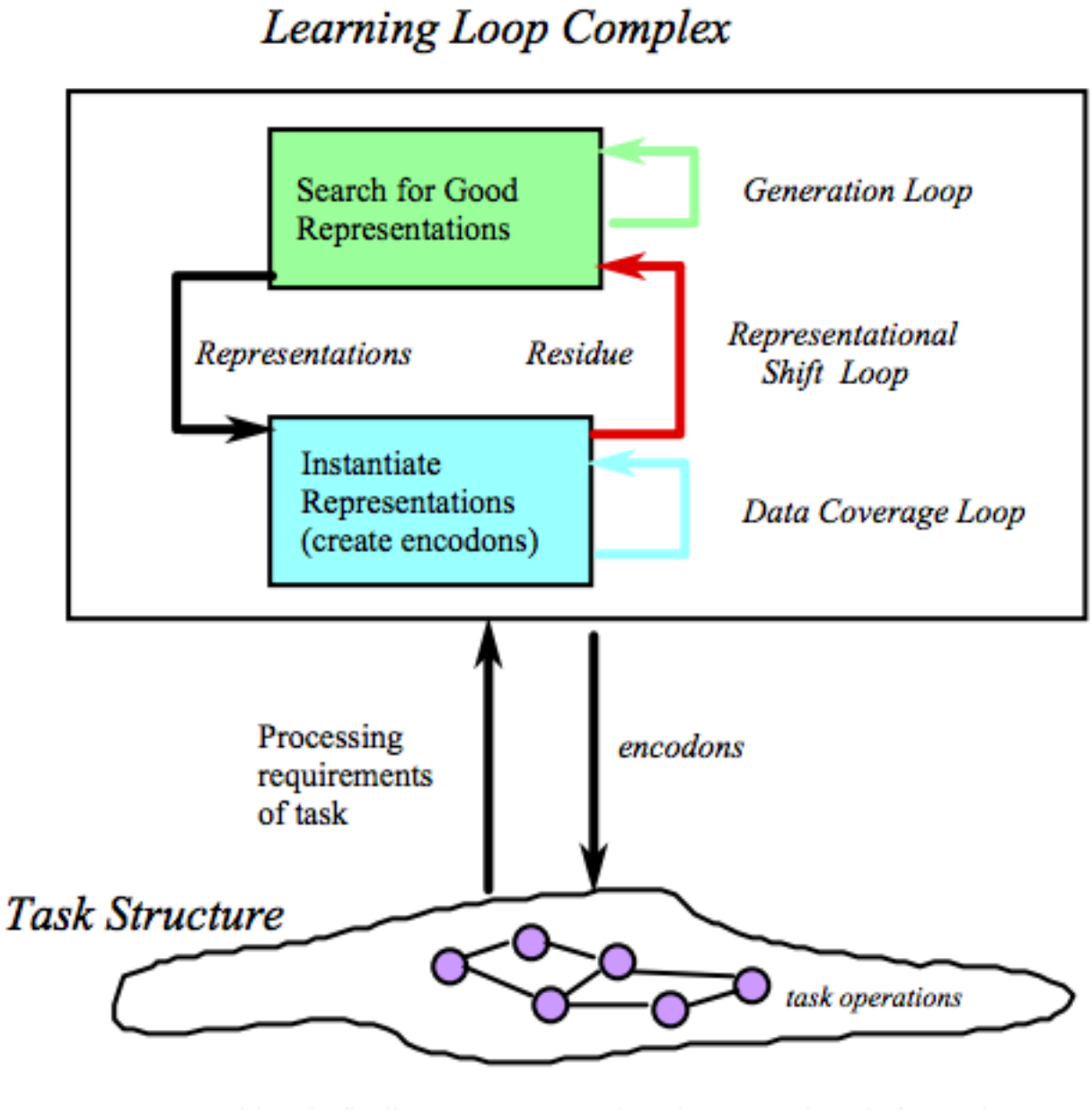


Figure 2.1: The Learning Loop illustrated by Russell et al. in "The Cost Structure of Sensemaking" [**?**]

representation matters is that if problem-solving can be framed as searching a problem space, then finding a compact representation of the problem reduces the size of the problem space that needs to be searched. Sensemaking has the potential to aid problem-solving by finding good representations. It does this by discovering structure within the data and establishing a representation that exploits the redundancy.

In the debate about whether a designer is necessary in the design process or whether it can be reduced to disembodied information processing, sensemaking claims that expertise matters. When developing schemas, there is a huge search space, and a brute force search would be exhaustive. Experts are needed to apply their knowledge and heuristics of the space to define the schemas. Only time will tell if expert knowledge can be removed from the system. At the present, sensemaking argues for a human-in-the-loop approach.

2.1.4 Design Patterns

Design patterns are a technique experts use for solving problems by reusing common patterns of successful past solutions. They are used extensively by designers because "[o]ne thing expert designers know not to do is solve every problem from first principles. Rather, they reuse solutions that have worked for them in the past" [**?**]. The patterns are general and abstract; they represent the "core of the solution" [**?**]. For example in software engineering, Model-View-Control (MVC) is a design pattern for interactive systems. Among other things it separates the model of the system from the view of the system, which allows the designer to provide multiple views of the same model. MVC provides a solution to a common software engineering problem. However, at the expense of being very general, it leaves many of the implementation details up to the designer.

Design patterns are used across many design fields: software [**?**], urban planning and architecture, [**?**] and web design [**?**], just to name a few. Abstract patterns are also used in purely creative fields like writing, although they do not refer to them as design patterns. The authors of the *Design Patterns* [**?**], the classic book on software engineering design patterns claim "Novelists and playwrights rarely design their plots from scratch. Instead, they follow

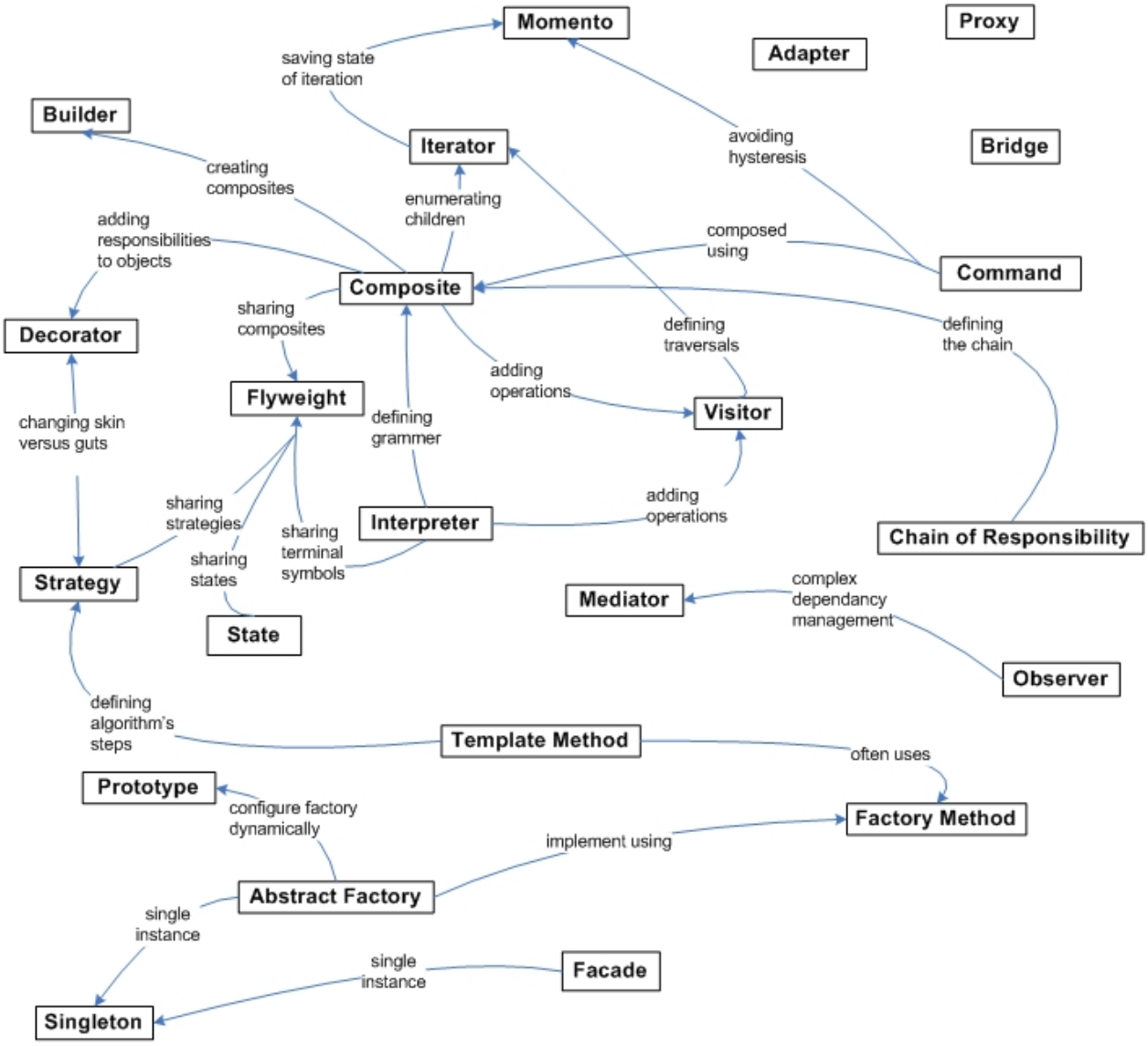


Figure 2.2: The complexity of relationships in software design patterns, from *Design Patterns* [?]

patterns like 'Tragically Flawed Hero' (Macbeth, Hamlet) or 'The Romantic Novel' (countless romance novels) [**?**]." The problems faced in all these domains are all open-ended – they don't have right or wrong solutions, just good or bad ones. Design patterns can help solve open ended problems because they present known good solutions to common problems. The design patterns that appear in books have been tested and honed over many years of expert practice [**?**]. They are not provably correct, but they are empirically validated as useful. Moreover, Design patterns are described abstractly enough that they can be molded to novel problems. If a designer can match their problem to a design pattern, then they are much closer to having a good solution. If not, they may be able to identify a subproblem that a design pattern can solve, which helps decompose the problem and makes progress towards solving it.

Design patterns are non-trivial to apply. They are abstract and need to be grounded in the context of the problem. There are many of them for a designer to pick from: 253 in the case of *A Pattern Language: Towns, Buildings, Construction* [**?**]. They span a large range of granularity and function. Design patterns are described with parameters that help designers apply them, such as: the problem they intend to solve, an example of a concrete instance of the problem the pattern solved, and related patterns. Determining which one to apply requires detailed knowledge of the patterns and the problem context. Moreover, the set of design patterns may have complex interrelations (See Figure **??**). Decomposing an open-ended problem into subproblems and applying the right design patterns to appropriate subproblems requires expertise. Even for experts, it is not a simple linear process. It is an iterative process that involves traversing a large graph of abstract patterns. Experts write books that describe the general problems the patterns solve, but it is up to the designer to match their problems with design patterns, and to adapt the patterns to their situation.

2.1.5 Background Summary

The prior work on problem solving, design, and sensemaking argue that there is a systematic approach to solving open-ended problems. For the time being, we still need human-level

intelligence for many of the steps. Fortunately, crowdsourcing is a way to embed human intelligence into software systems.

2.2 Crowdsourcing

Crowdsourcing is a term coined in 2006 by Jeff Howe [**?**] to describe the emerging practice of accomplishing a task by asking a large crowd of people to do it rather than hiring employees. Crowdsourcing is typically mediated by an online platform to connect the crowd and collect the contributions. The platforms vary widely in the scope, motivation, and degree of expertise needed for the tasks [**?**]. In particular, the tasks can be large complex tasks that require expertise such as writing an encyclopedia [**?**], or small, easy microtasks like labeling images [**?**].

This dissertation is only concerned with microtask crowdsourcing. Microtasks are small tasks with low contextual overhead for workers. They are exciting to computer science because microtasks have the potential to become building blocks in an otherwise traditional computer program that can give computers an escape hatch to human-level intelligence when they need it. Microtask crowdsourcing has been essential as a method of collecting large data sets for computer vision [**?**, **?**, **?**]. It can be done in real-time to answer questions for blind people [**?**], to control robots [**?**], transcribing audio [**?**], and having a conversation [**?**]. Communities can be engaged to participate in crowdsourcing tasks that require expert knowledge such as grading computer science exams [**?**], engaging in political debate [**?**], and conducting biology research [**?**, **?**]. Workflows can be written to ensure quality [**?**, **?**] and to combine microtasks into more complicated artifacts [**?**, **?**, **?**, **?**]. Naive workflows can be optimized for cost and quality [**?**, **?**]

In this section, we discuss the historical background of crowdsourcing, the rise of microtask crowdsourcing on the web, the advent of crowd algorithms, the progression toward more complex crowd algorithms, strategies for decomposing problems and reassembling microtask results into a coherent whole, and the need for adaptive crowd algorithms,

2.2.1 Historical Microtask Crowdsourcing

There are examples of microtask crowdsourcing scattered throughout history. The Math Tables [**?**] project during the Second World War broke complex calculations into subtasks such as addition and subtraction and used one dedicated person for each type of task, who then passed their work to the next person when they finished with their subtask. Dividing the work into subtasks and creating a physical workflow around them allowed these teams of "computers" to calculate faster, more reliably, and to perform a function that perhaps some of the people could not perform alone. Today, for mathematical calculations, we no longer need humans to be the computers. Computers can represent mathematical problems and algorithms can operate on numbers better than most people. The problems that computers cannot solve but people can involve human perception, world knowledge, common sense and and language abilities that computers do not yet have. These tasks also tend to be more subjective than mathematical tasks. These subjective tasks are a new realm where crowdsourcing can help computers.

2.2.2 The Rise of Microtask Crowdsourcing: Games with a Purpose

The ESP Game [**?**] was a breakthrough in modern microtask crowdsourcing. However, it did not describe itself as crowdsourcing. It described itself as a "game with a purpose" (GWAP) that used "human computation" to do one task computers could not do: image labeling. At the time, internet image search quality was low because computers could not see what was in images and surrounding text did a poor job of describing images. The ESP Game collected and verified images labels from people that was used to help improve image search. Its interface was a website where volunteers were paired with an anonymous partner to play a game. They were shown an image and they entered labels for it. The label was only accepted if both players entered it independently. Independent agreement between players on labels ensured that the label was accurate. The game was successful in collecting millions of labels and attracting thousands of players in large part because it was fun. It has smooth

game mechanics, a low start-up cost due to simple instructions, and playing cooperatively with a partner online made it appealing.

The authors of the ESP Game and a few other researchers with backgrounds in games succeeded in creating other games with a purpose-style crowdsourcing websites. Other games with a purpose include TagATune [**?**] for tagging music clips, Search War [**?**] for web search relevance judgments, as well as graphics-heavy citizen science projects such as Fold.it [**?**] and EteRNA [**?**]. Other citizen science sites such as Galaxy Zoo [**?**] for categorizing images of galaxies were also fun and popular, although not explicitly gamified. Despite this success in crowdsourcing data for free, many researchers and practitioners found the overhead required to make an interface fun and to attract a crowd was too large and risky to scale to many problems.

2.2.3 Paid Microwork: Mechanical Turk

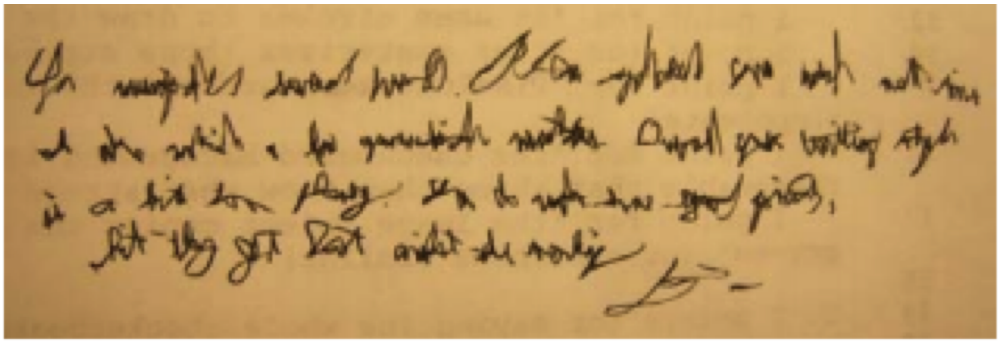
Inspired by the ESP Game, Amazon.com released a paid microwork platform, Mechanical Turk [**?**] – an online labor market where requestors offer workers small amounts of money for quick tasks such as labeling images or classifying products into categories. Interfaces tend to be basic, and workers were motivated by money rather than fun. Thus the overhead for requesters to crowdsource data collection was significantly reduced. However, this platform was still geared toward simple independent tasks performed in parallel.

2.2.4 Iterative Microwork

A first step towards complex crowdwork was TurKit [**?**], a toolkit I helped design and build for running iterative tasks on Mechanical Turk. For iterative tasks, the work of one worker was given to another worker to iteratively improve the work. For example, in creating an image description, one worker could improve upon another worker's image description by fixing typos, and adding new details (See Figure **??**). Several workers could do this iteratively, continually improving the image descriptions until they were of sufficiently high quality.

Iteration 1: Lightening strike in a blue sky near a tree and a building.
Iteration 2: The image depicts a strike of fork lightening, striking a blue sky over a silhoutted building and trees. (4/5 votes)
Iteration 3: The image depicts a strike of fork lightning, against a blue sky with a few white clouds over a silhouetted building and trees. (5/5 votes)
Iteration 4: ~~The image depicts a strike of fork lightning, against a blue sky- wonderful capture of the nature.~~ (1/5 votes)
Iteration 5: This image shows a large white strike of lightning coming down from a blue sky with the tops of the trees and rooftop peaking from the bottom. (3/5 votes)
Iteration 6: This image shows a large white strike of lightning coming down from a blue sky with the silhouettes of tops of the trees and rooftop peeking from the bottom. The sky is a dark blue and the lightening is a contrasting bright white. The lightening has many arms of electricity coming off of it. (4/5 votes)

Figure 2.3: TurKit's Improve-and-Vote algorithm iteratively improved image descriptions.



Iteration 1:

You (?) (?) (?) (work). (?) (?) (?) work (not) time. I (?) (?) a few grammatical mistakes. Overall your writing style is a bit too (phoney). You do (?) have good (points), but they got lost amidst the (writing). (signature)

Iteration 6:

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)

Figure 2.4: The crowd could decipher messy handwriting by having each worker build on the contextual clues of other workers.

TurKit gave rise to the notion of workflows built from microtasks. These workflows could produce higher quality work than any one worker could produce. In the case of deciphering the words in blurry text or bad handwriting, the iterative approach could solve problems that no single worker could solve by himself (See Figure **??**). In addition, workflows can be optimized to increase quality and minimize the number of steps necessary to achieve it [**?**].

The main crowd algorithm used in TurKit was Improve-and-Vote, which involved two types of microtasks: to "improve" something (such as a caption for as image), or to "vote"

on whether one artifact was better than another. This simple algorithm was useful for tasks such as writing image descriptions and deciphering blurry handwriting, but was too simple to crowdsource large artifacts like a Wikipedia page. Improve-and-vote is limited to small tasks because it not decompose the task into smaller, easier subtasks. A new worker tasked with improving the current output would have to be aware of the entire context of the problem, and decide what improvement to make. If no worker were willing or able to understand the entire context and improve it, the Improve-and-Vote algorithm would fail. In order to crowdsource larger artifacts, tasks must be decomposed into subtasks.

2.2.5 Decomposing Tasks into Subtasks

Soylent [**?**] is a system implemented on top of TurKit that uses crowds to edit Microsoft Word documents. Soylent decomposed editing tasks into a three-step workflow called Find-Fix-Verify. To make edits such as shortening text, fixing typos, or inserting images, crowd workers were solicited to first find places in the document that need attention, then fix the text in need of improvement, and lastly verify edits. The Find-Fix-Verify workflow was more sophisticated than Improve-and-Vote in part because it decomposed the task of editing into easier subtasks. Decomposing the editing task reduces the workload and responsibility of an individual worker and increases overall work quality by having more opportunities to check the work. For example, before fixing a problem in the text, multiple workers are redundantly asked to find problems. Redundancy ensures that there is agreement on the problem areas before they are fixed.

Soylent's Find-Fix-Verify decomposes tasks more than TurKit's Improve-and-Vote algorithm, but the work that can be done with Find-Fix-Verify is limited in its complexity. Find-Fix-Verify can only make local edits that do not require global document context. Fixing typos or removing unnecessary words can be done independent of the rest of the document. For these types of editing tasks, there are no complex interdependencies and no side effects. For more complex tasks such as changing the running example through the document, tasks are likely to have complications that are difficult to foresee. The linear workflow only work

when there are no unanticipated complications with the task. For more complicated tasks such as content creation and data organization, workflows will need to be more flexible and adaptive.

2.2.6 Strategies for Complex Crowdsourcing

There have been a few crowdsourcing systems that address the challenge of complex tasks. Each one has a different perspective on how to decompose tasks and reassemble the results into a cohesive whole.

Unstructured Approach

One way to decompose a problem is to let the workers define the subtasks. This is the approach of the Turkomatic system [**?**]. Turkomatic's process to complete a large task such as writing a piece of software is to ask one worker to either do the task, or chose to break the task into subtasks. This repeats recursively until all the subtasks are completed. Although this solution is elegant and fits well with the traditional algorithmic approach of divide and conquer it had three fatal flaws. First, tasks are hard to decompose. Particularly, subjective and complex tasks, such as writing software, are difficult to decompose. It is not reasonable to expect a worker to be able to decompose a large complicated task. Second, because workers had only partial context for their decisions, and did not have a global overview of the plan, they could easily create tasks that had already been created or do conflicting work. Third, it was not always obvious how two tasks could be put back together into a cohesive whole. These are all problems that are traditionally faced in non-co-located collaboration, even outside of crowdsourced microwork. This unstructured approach was deemed unwieldy and too error prone. A more structured approach would be needed to decompose complex tasks.

Structured Approach

The CrowdForge [**?**] system took a highly structured approach to crowdsourcing workflows. CrowdForge allows the requester to decompose the task and define the algorithm to reassemble the results. CrowdForge introduced a workflow inspired by MapReduce [**?**] for the complex task of creating blog posts. Traditional MapReduce is an distributed systems model developed by Google to massively parallelize and distribute tasks such as counting word instances in web pages. The "map" step sends documents to many different machines, which each machine create a list of word counts independently. Then in the "reduce" step, all the individual word count lists are aggregated to one machine and combined. Combining hundreds or word counts lists is a fairly trivial data processing task with no complex interdependencies. In CrowdForge, the "map" step was used to gather various pieces of information needed for the blog post and the "reduce" step was used to merge them back together. This workflow was successful is creating blog posts and showed that structured approaches are promising for some problems.

An issue with these structured, process-centric approaches is that they are not sensitive to the needs of the particular problem or context-specific issues that might arise. If MapReduce is used only for numerical operations such as counting words, no extra context is going to arise. However, most subjective and complex problems have unique aspects that must be dealt with individually, and cannot be accounted for by an inflexible process. Additionally, nothing guarantees that the data produced by these processes can be merged back together cohesively. This points to the notion that a more adaptive approach might be necessary for some problems.

A follow-up to the structured approach was a workflow for creating inventions, similar the ones sold on Quirky.com [**?**]. Quirky.com has a range of simple inventions, many of which are household products such as a rake that pushes leafs off more easily. Yu et. al. [**?**, **?**] used the crowd to generate Quirky.com-type invention ideas through the extraction and application of schemas. The crowd looks at examples of past inventions and extracts schemas, for example,

"Attach X to Y to keep X from moving." Then applies these schemas to new domains by finding new potential values for X and Y. The insight is that the solution schemas that were used to solve one problem could possibly be used to solve another, using abstraction and analogy-based reasoning. With this approach, the system was able to extract schemas from the crowd and to apply them in new domains to discover new ideas. The ideas were evaluated on their novelty, but not implemented or sold to test their physical and economic practicality. The structure of schemas, and particularly, analogical reasoning has many applications.

Unfortunately, most complex problems cannot be solved by applying a single schema. Complex problems, such as writing software, are benefitted by applying design patterns [**?**, **?**] which are similar to schemas in that they present abstract solutions to common problems. However, to solve a large problem, often multiple design patterns must be applied and it is non-trivial to pick and apply the right design pattern to the right problem.

Constraint Satisfaction Approach

One way to add structure to the unstructured approach is to use global constraints. In the Mobi system [**?**], workers were given the task to create a one-day schedule for a person visiting a new city. The schedule had many global constraints, such as "must visit at most one museum," "must visit at least two coffee shops," and "must eat at vegan restaurants only." The challenge for the workers was to create a schedule that met these global constraints. No constraint solvers were used – all the constraints were met by workers aided by the Mobi user interface. Formulating a creation problem in terms of global constraints gives some high-level guidance to the task that helps align the efforts of independent workers. A rich map-based user interface helped workers visualize the entire context and what needed to be done next.

Similar to Turkomatic, workers could create subtasks within the interface by generating "todo items." Workers could choose between doing a todo item or adding new todo items. In contrast to Turkomatic, Mobi successfully created outputs with global cohesion because all the workers were aware of the global constraints. Although the worker did not communicate

to each other directly, and were working independently in serial, "todo items" became a structured way of communicating to future workers steps necessary to achieve a cohesive goal without undoing each other's work. The evaluation showed that workers were able to complete the task and meet the constraints. In an ablation study, the authors of Mobi showed that the interface with todo items completed significantly faster than in a version of the interface without todo items.

Mobi was substantially more successful than Turkomatic. The semi-structured approach fixed some of the problems that Turkomatic had, such as the problem that tasks were de-contextualized, and thus not decomposed in a way that that made it feasible to reassemble the results into a cohesive whole. However, it is unclear that the technique Mobi used would scale if the whole context of the problem did not fit into one user interface, or were easy to visually understand the entire context of. Like TurKit, Mobi's iterative approach requires workers to have access to the global context, which limits the size of the problem that can be solved.

The quality of the itineraries produced with Mobi were generally good in that they fit the constraints posed by the requester. However, the schedule was completely full. The crowds were eager to meet the constraints and blindly worked towards filling the schedule. However, an expert reflecting on the schedule might have recognized that the schedule was exhausting. Leaving time for rest breaks is common sense that a travel agent might have but that a crowd blindly following an objective function would not. This is the type of constraint that is not obvious upfront, but appears dynamically as the solution emerges. This is an example of *tacit knowledge* - common sense rules that experts easily recognize when they see violated, but never write down because there are so many. With most complex problems, the subjectivity and dynamic nature of the problem allows for unwritten rules that need to be obeyed to solve the problem. This is perhaps the greatest issue with the constraint satisfaction approach to solving more complex problems: absolute coherence to written constraints without the ability to dynamically discover and adapt to new constraints.

2.3 Groupware

Groupware is software that is designed to enable multiple people to collaborate on a common task to achieve a goal. Examples of groupware include email, calendaring systems, and other office software that enables communication and collaboration between people. Groupware is relevant to crowdsourcing in that they have the same goal: collaboration towards a goal. However, they implemented quite differently. Groupware does not traditionally contain workflows, or divide work into microtasks. It has a more open framework for collaboration. Likewise, it models its workers as collaborators, not workers. There are lessons that crowdsourcing can learn from groupware.

Groupware has a long history in HCI and has proven to be unexpectedly challenging. In analyzing several expensive failures of groupware systems, Grudin [**?**] observes that developing for groups is more difficult than developing for individuals and he supplies eight challenges of developing groupware systems. Many of these challenges do not apply to crowdsourcing. For example, a danger in groupware is that it does not attract a critical mass of users to be useful. This is less of a problem in crowdsourcing where workers motivated to work by payment. Groupware can have a "disparity in work and benefit" where users who have to do the work do not necessarily get a benefit from it. In crowdsourcing, paying workers by the unit of work alleviates this problem. However, one groupware challenge does apply to crowdsourcing: that many of the tasks we build crowdsourcing systems for are replacing an existing social process that we do not complete understand.

Social processes are complicated, not well understood, and unexpectedly difficult to replace with software systems. Often, the people designing collaborative systems do not fully know all the important details of the system they are replacing. They may know the stated rules for how the current system is supposed to work, but they have not experienced or observed the numerous and important cases of *exception handling*, which are not embedded in the rules. As Grudin describes: "Work processes can usually be described into ways: the ways things are supposed to work and they way they do work. Software designed to support

standard practices can be too brittle... groupware that enforces 'standard procedures' can bring work to a halt." The way systems do work relies on the flexibility of people to enforce the spirit of the law rather than the letter of the law. Often this knowledge about which rules can be bent and how to get around them is *tacit knowledge*. People doing their jobs have picked up on these rules and practices, but they do not write them down because they are many of them, and its hard to describe the exact circumstances that they apply. But experts know the circumstances when they see them.

It can be impossible to create workflows that can replace social processes governed by tacit knowledge. Instead of rigid workflows, more flexibility is needed. One way to increase flexibility is to use constraints, but allow workers to meet them flexibly. The system can help workers achieve global constraints, like Mobi, but there must also be flexibility in meeting softer constraints that arise. The Cobi [**?**] system for collaborative conference scheduling allows global constraints to be imposed by the system and soft constraints to be added as they arise. Frenzy [**?**] builds on this work to collaboratively crowdsource the creation of sessions that need to be organized.

2.4 Computer Science Approaches to Open-Ended Problems

The systems in this dissertation address two kinds of open-ended problems: data organization and the creative task of writing humor. Previous systems have approached these problems with systems that are either single-user, multi-user, crowdsourced, partially automated or fully automated. The follow subsections describes previous approaches.

2.4.1 Clustering and Taxonomy Creation

Machine learning algorithms, such as LDA [**?**], can automatically cluster data. While these automated approaches are fast and cheap compared with human labor, they have several limitations. First, machine algorithms require a human to explicitly specify a set of features for each media type, such as the bag-of-words representation for text or computer-vision features for images. Second, their performance has short-comings that are obvious to human

judges – the output often creates clusters that are incoherent to people. Finally, machine algorithms have trouble naming the resulting clusters or categories; they are incapable of creating an overarching abstractions, they can only identify a common feature such as a word or phrase in the case of text.

Several machine-learning techniques have been refitted to use human inputs. CrowdKernel [**?**] learns an SVM kernel from human pairwise comparisons. Matrix completion has been used to do clustering with spare pairwise labels [**?**]. Additionally, discriminative labels [**?**] can be found for image datasets using a mixture of probabilistic principal component analyzers (MPPCA). Crowdclustering [**?**] uses human-created clusters of a subset of the data and Variational Bayes method to create unlabeled clusters of data and subclusters. These are novel and intriguing ways to combine human and machine effort to complete labeling, and find clusters in data. However, what was still needed is an end-to-end system that requires only quick, easy tasks for workers and produces human readable category labels on a taxonomy organizing all the data. Cascade [**?**] was the first such approach and Community Clustering [**?**] was another.

There are several distributed human taxonomy creation efforts. First, Wikipedia maintains a taxonomy of all its pages [**?**]. The work is coordinated though talk pages, not through a crowd algorithm. Although this is a success in taxonomy creation, it requires a large time commitment overhead and thus it is not easy for individuals to contribute in small increments. Second, card sorting [**?**] is a technique for members of a group to contribute to an organization of their data. Participants create labeled clusters for all data and the similarities between participants' clusters tells the moderator how people mentally cluster the space. Card sorting is an investigative technique. It is not designed to produce a usable categorization. Third, folksonomies are community-created labeled clusters of data. WordNet – a database of human-contributed semantic knowledge of links between works – has been used to turn folksonomies into hierarchies of concepts [**?**]. It seems that human common sense and linguist ability are currently necessary for taxonomy creation. Thus, the Cascade algorithm presented in this dissertation uses these human abilities and breaks down the task

of creating a taxonomy into short, easy units of labor.

2.4.2 Applying Design Principles and Design Patterns to Support a Creative Process

Human-computer interaction (HCI) recognizes the importance of design and and the human-centered design process. Many HCI researchers have built systems that support the design process, particularly through prototyping tools [**?**, **?**, **?**]. Before building tools to support designers, they observe and interview designers to understand their current process. For example, Newman et al. observed 11 website designers to understand the problems associated with web design [**?**]. Informed by this studies, the authors built tools to support the open-ended task of website design such as supporting multiple concurrent representations [**?**]. HumorTools is essentially a design tool for humor novices and thus we wanted it to be informed by expert practice. Unlike physical design and even software design, creating humor is a largely mental and subconscious process, which makes it hard to observe. Although we cannot observe the subconscious humor creation process, we can still learn from experts' description of their process and from back-engineering examples of humor.

There has been and impressive set of systems that take design principles expressed by designers and encode them into software systems such that design can be completely automated. LineDrive encodes rules about the readability of maps to automatically produce intuitive and readable maps to accompany driving directions [**?**]. The design principles of interior design were encoded into a system that automatically produced aesthetic and functional room designs [**?**]. The design principles of cut-away diagrams were encoded to produce clear images to accompany step-by-step assembly instructions [**?**]. All these systems have a general principle in common: they automate design by using the design principles to define a search space and optimization function, then solve the design by searching the space with an algorithm such as gradient descent in order to maximize the optimization function [**?**]. HumorTools also poses a design/creativity problem as a search problem. However, systems such as LineDrive solve visual-spatial problems which can be easily represented by a numerical search space - for example, the optimal angle between two lines in a map or the optimal

(x,y) position of two chairs in an interior design blueprint. In contrast, HumorTools searches a conceptual space of associations. Numerical spaces are currently easier to search with traditional optimization algorithms. For conceptual spaces, we still need human intelligence.

Design patterns have long been known to be important in design but we still need human intelligence to apply them. Motif [**?**] is an example of a system that uses expert narrative patterns in film to help novice film makers structure their home videos. The authors extracted narrative patterns from expert examples. Users of Motif could pick the design patterns that best suit their video and their taste. The only guidance users received was being told to use at least one pattern from each of the three types of patterns: beginning patterns, middle patterns, and end patterns. This light structure with many options helped user create better, more cohesive home videos than the control condition.

Design, design principles, and design patterns are powerful tools. Whether they are employed by people or by machines or by a combination of the two, they are key to decomposing creative tasks. In HumorTools, we use microtasks to encode comedic design patterns and guide novices in the HumorTools workflow.

2.4.3 Crowdsourcing Creativity

Crowdsourcing has attempted to automate creativity in limited but interesting ways. Yu and Nickerson [**?**] crowdsourced the design of chairs by mixing ideas across users to spur innovation. Yu and Kittur [**?**, **?**] used the crowd in a two-stage, analogy-based product idea generation. This approach has two stages: 1) using the crowd to extract analogy patterns, or schemas such as "use x to affix to y". Step 2 applied these patterns to new cases. Analogy is a powerful technique for ideation and creativity, and a technique that people are still better at than computers. However, the use of analogy is exciting, but additional techniques such as deep understanding of problems and empathy are other techniques from the design literature could be added.

Crowdsourcing and microtasks have been valuable to help evaluate and understand creative artifacts. Voyant [**?**] and CrowdCrit [**?**] allow users to upload visual designs and get

feedback from the crowd. Feedback can range from first impressions to what visual elements drew the user's eye, and can be structured based on the types of feedback that follow design principles.

In the space of humor, microtasks were used to help non-native English speakers understand humorous memes in English by annotating the memes according to Semantic Script Theory of Humor [**?**]. Understanding and evaluation is an important part of the design process and it is clearly a problem where getting fresh perspectives and suggestions from the crowd is useful. However, evaluation is just one step in the overall process of creating novel and useful artifacts.

2.4.4 Computational Humor

Artificial Intelligence researchers have long studied humor as a test of computer intelligence, language understanding, and creativity, but most efforts have aimed at classifying what is and is not humorous. A 2015 effort to classify submissions to *The New Yorker* Caption Contest had a 69% success rate using deep learning with human labeling of jokes and captions to add context needed for computers to understand the jokes and images [**?**]. This is useful for helping the contest organizers sift through thousands of submissions. Other researchers were able to classify which knock-knock jokes were funny based on presence of word play[**?**]. Other efforts focused on the problem of detecting sarcasm [**?**] with some success.

To date there have been very few effort at computational humor generation. The JAPE punning-riddle generator [**?**] is one of the first attempts of computational humor. It looked at both double meaning and close rhymes and found ways to fit them into question-and-answer joke format. Here are two examples of jokes that JAPE generated:

Q: What is the difference between leaves and a car?
A: One you brush and rake, the other you rush and brake.

Q: What do you call a strange market?

A: A bizarre bazaar.

JAPE relies heavily on puns, a form of word play that is especially tractable for computers, because word meanings are already annotated in dictionaries and understanding them requires minimal context or world knowledge. However, word play is a small subset of humor, and some of the funniest humor does rely on world knowledge and makes fun of the general human experience.. Kiddon and Brun were able to create jokes by detecting another form of word play, double-entendres [**?**]. Given a sentence, they trained a word- and phrase-based model to detect if there would be double meaning if somebody said "That's what she said" after them. For example, a sentence containing "banana" is more likely create a double-entendre opportunity than a sentence containing "door." Their approach has an impressive 71% accuracy, but is in essence still just a classifier, not a general humor-generation strategy. Despite this progress on special types of humor, there is a significant amount of human input is still necessary in any method for generating humor that goes beyond word play and moves towards more general approach to creating humor.

Chapter 3

CASCADE: CROWDSOURCING TAXONOMY CREATION

Taxonomies are a useful and ubiquitous way of organizing information. However, creating organizational hierarchies is difficult because the process requires a global understanding of the objects to be categorized. Usually one is created by an individual or a small group of people working together for hours or even days. Unfortunately, this centralized approach does not work well for the large, quickly-changing datasets found on the web. Cascade is an automated workflow that creates a taxonomy from the collective efforts of crowd workers who spend as little as 20 seconds each. We evaluate Cascade and show that on three datasets its quality is 80-90% of that of experts. The cost of Cascade is competitive with expert information architects, despite taking six times more human labor. Fortunately, this labor can be parallelized such that Cascade will run in as fast as five minutes instead of hours or days.

3.1 Introduction

Although taxonomies are a useful and ubiquitous way of organizing information, creating these organizational hierarchies is difficult because the process requires a global understanding of the objects to be categorized. Currently, most taxonomies are created by a small group of experts who analyze a complete dataset before identifying the essential distinctions for classification. Unfortunately, this process is too expensive to apply to many of the user-contributed datasets, e.g. of photographs or answers to questions, found on the Internet. Despite recent progress, completely automated methods, such as Latent Dirichlet Allocation (LDA) and related AI techniques, produce low-quality taxonomies. They lack the common sense and language abilities that come naturally to people.

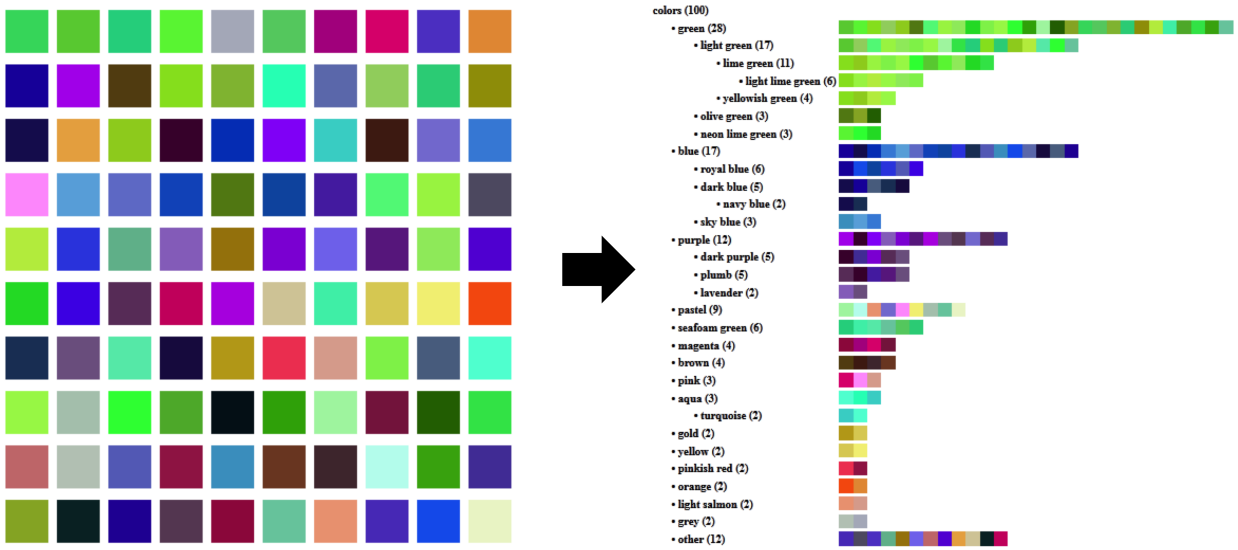


Figure 3.1: Example input and output of Cascade. The input is 100 random colors; the output is a taxonomy of the colors.

This paper presents Cascade, a novel method for creating taxonomies. Cascade is a crowd algorithm that coordinates human labor with automated techniques. Each worker need only make a small contribution with no long-term time commitment or learning curve. None of the workers needs to have a global perspective of the data or the taxonomy under construction. Figure **??** contains example results.

Crowdsourcing has become a popular way to solve problems that are too hard for today's AI techniques, such as translation, linguistic tagging, and visual interpretation. Most successful crowdsourcing systems operate on problems that naturally break into small units of labor, e.g., labeling millions of independent photographs. However, taxonomy creation is much harder to decompose, because it requires a global perspective. Cascade is a unique, iterative workflow that emergently generates this global view from the distributed actions of hundreds of people working on small, local problems.

In this paper we first describe the lessons learned from early, failed attempts to design a crowd workflow for building taxonomies. We next present the Cascade algorithm and the

three human intelligence task (HIT) primitives used to implement it. We demonstrate the results of running Cascade on five representative data sets. We evaluate Cascade in three ways: we compare its time and cost to that of four expert information architects who were paid to taxonomize the same data, we count the number of mistakes in Cascade?s output and interpret it as an error rate, and we compare the coverage of the categories in Cascade to those of the expert-made taxonomies.

This system makes the following contributions:

1. We present Cascade: a novel crowd algorithm that produces a global understanding of large datasets. Cascade is an online algorithm that can update the taxonomy as new data arrives. The tasks given to workers are quick and parallelizable.

2. We propose three HIT primitives, simple task interfaces, used as building blocks by Cascade and useful for future crowd algorithms.

3. We introduce Global Structure Inference as a way to combine independently-generated judgments into a cohesive taxonomy.

4. We evaluate Cascade on three datasets showing that Cascade can perform close to expert level agreement (80-90% of expert performance) for a competitive cost.

3.1.1 Initial Approaches

The Cascade algorithm evolved from a sequence of initial prototypes. Our experience resulted in several surprising observations that informed our ultimate design of a crowdsourced taxonomy workflow.

Iterative Improvement

Iterative improvement is a general crowdsourcing pattern first described in TurKit [**?**]. Iterative improvement has proven successful at using multiple workers to build on and improve

each other's image descriptions, and to collectively decipher blurry text or bad handwriting. CrowdFlower and Karampinas, et al. [**?**] have applied iterative workflows to clustering. Although it is possible for this method to work, our experiences were largely negative. We applied iterative improvement to taxonomy creation by giving workers a list of items and an editable hierarchy interface. Workers were asked to improve the taxonomy by adding, deleting, or moving categories or by placing items in the taxonomy (Figure **??**). We observed that the iterative improvement interface suffered from two main problems:

1. The taxonomy grows quickly, making the tasks more time consuming and overwhelming as time goes on. Creating the first category and placing a few items in it is quick and easy. However, when 50 categories must be read in order to figure out whether or not an item belongs in any of the existing categories, the work becomes so challenging that single workers have a hard time making contributions in a short time frame.

2. Second, the task had many options for how to contribute (add categories, place items, merge categories, etc.) and workers had trouble selecting tasks. Although giving workers options for how to contribute made the task very flexible, it also meant workers had to decide what was important to do next. It was not always obvious what meaningful work should be done or how long it would take. A similar result was found in the Mobi system [**?**]. We concluded that we needed to break down the taxonomization task into manageable units of work and have an algorithm coordinate the work.

Category Comparison

Similar to the ESP Game [**?**], we found that workers are eager and willing to suggest category labels for data. The problem is deciding which labels are useful. The final taxonomy should have a non-redundant set of categories with parent-child relationships. We tried many approaches for directly soliciting these relationships. Figure **??** shows one of our prototypes: a matching game where workers can select pairs of related category labels.

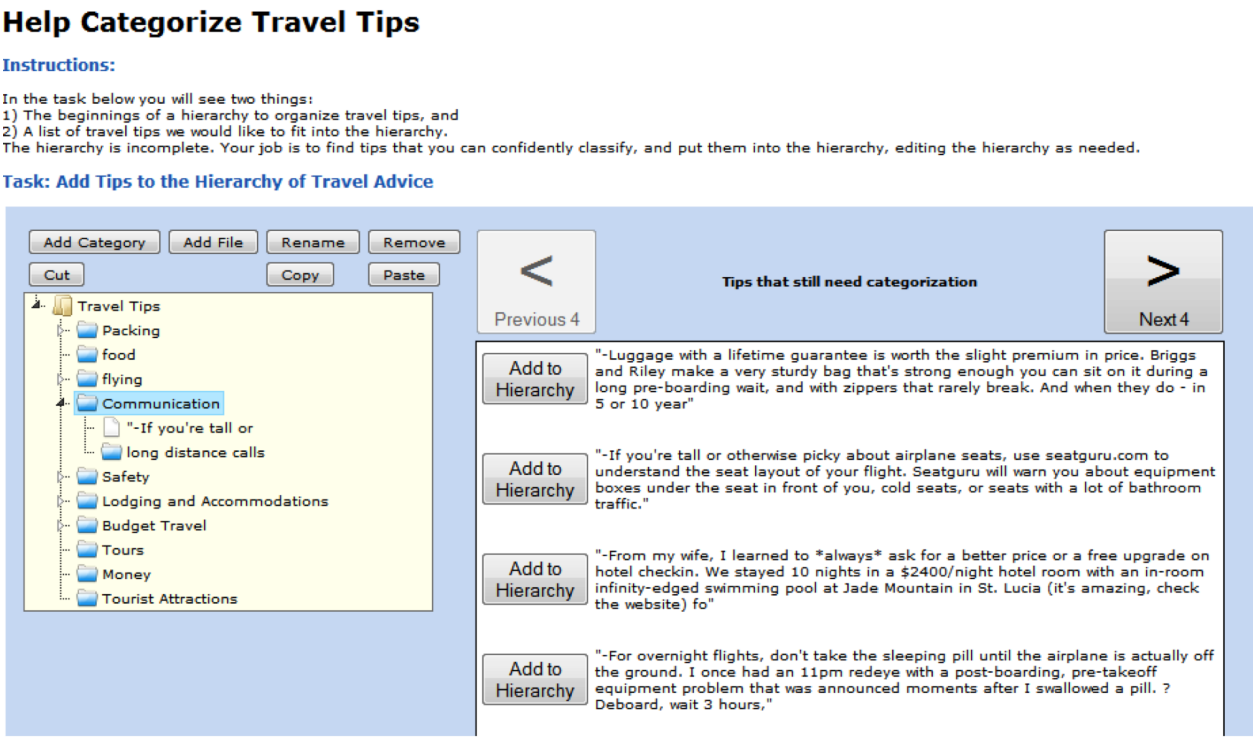


Figure 3.2: Early prototype interface: iterative improvement

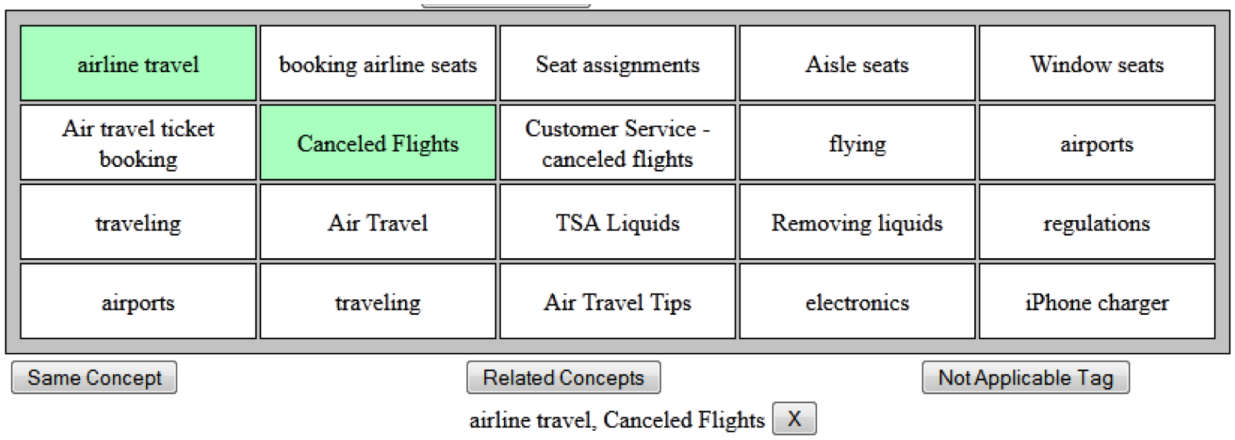


Figure 3.3: Early prototype interface: category comparison

We found that workers were wildly inconsistent in the relationships they saw in the category labels. For example, some thought "air travel" was the same as "flying," while others thought it was a superset. Some thought "packing" was the same as "what to bring," while others thought "what to bring" was a subset of "advice" but "packing" was not.

We concluded that it was a mistake to ask workers to compare abstractions with abstractions. The differing assumptions people make about abstractions are too hard to write down, and they render individual worker judgments incomparable. Instead, judgments should be made relating actual abstractions to data: in this case, relating categories to items.

Clustering

In order to elicit both clusters and cluster names from workers, we prototyped an interface that presented workers with a small number of items (8-10) and asked them to suggest categories that fit at least two items (Figure **??**). Although workers found the task easy and intuitive, the quality of the categories was not as good as when we generated category suggestions for single items. Restricting workers to naming categories that satisfied multiple items encouraged workers to name overly broad categories. They often gave categories names

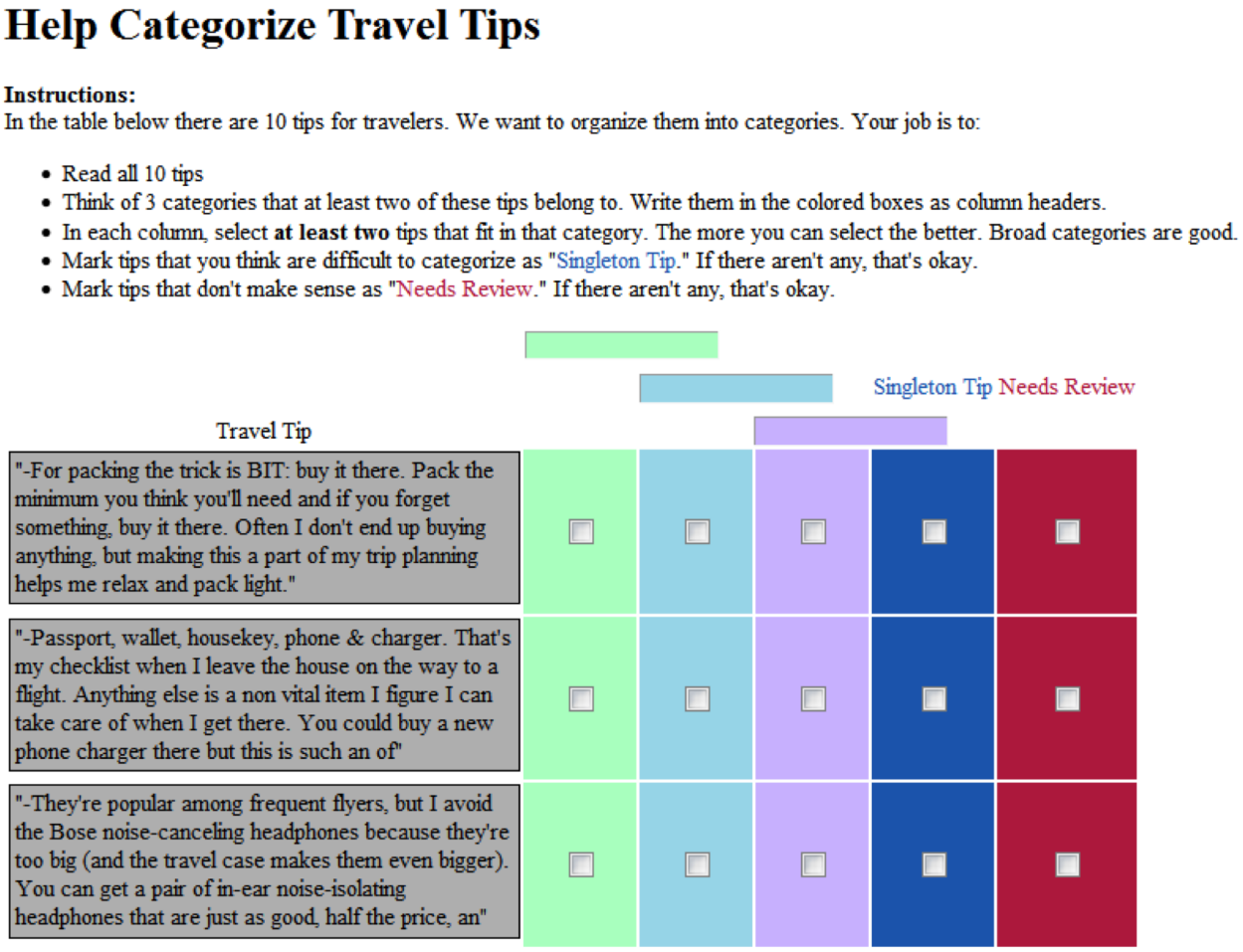


Figure 3.4: Early prototype interface: item clustering

such as “good tips” or “advice” because they would fit multiple of the 8 items displayed. However, these are not useful category names. Moreover, what workers wanted to do instead was name categories they felt were good even if they only fit one item. Workers know these categories are good because of their knowledge of the world. For example, workers can use their common sense and intuition about travel advice and infer that “TSA Security” might fit multiple items even though it only fits one item in the 8-item subset.

We decided that the clusters workers found in a small subset of the data were often unnatural and forced, and that it was better to allow workers to suggest categories that fit

one item very well rather than fit multiple items more loosely.

3.2 The Cascade Algorithm

We now discuss Cascade's inputs and outputs, the parameters governing its execution, its three HIT primitives, and the flow of the algorithm itself.

3.2.1 Inputs

Cascade takes two inputs: a set of items to be categorized and a descriptive phrase (the topic) identifying these items. An example of inputs to Cascade are 100 responses to the question "What is your best travel advice" with the topic "Travel Advice." In theory, Cascade can handle item-sets of arbitrary size, but the algorithm's expense grows super-linearly in the number of items. To date we've tested on five datasets, ranging in size from 22 to 200 items. In the future, we plan to use co-occurrence statistics to optimize the cost on large inputs.

3.2.2 Output

The output of Cascade is a taxonomy consisting of labeled categories and associated items. More precisely, Cascade generates a tree whose nodes are labeled with a textual string, called a category; the tree's root is labeled with the topic input, and an "other" node is added as a child of the root if necessary. Items may appear in multiple categories in the taxonomy; for example, a travel tip about flight deals may appear in "air travel" and "saving money."

3.2.3 Parameters

Cascade's behavior is guided by a set of parameters. Default values which were used in our experiments are noted where applicable.

- Let n be the number of items input.
- Let $n \leq n$ be the number of items considered in Cascade's *initial item-set*, default = 32.

- Let $n' = n\text{-}m$ denote the number of items in the *subsequent item-set*.
- Let k be the *replication factor* (the number of workers who may be asked to repeat a step), default = 5.
- Let g be the first voting threshold, default = 2.
- Let h be the final voting threshold, default = 4.
- Let t be the maximum number of items shown to a worker at once, default = 8.
- Let c be the maximum number of categories shown to a worker who is selecting the best category for an item, default = k = 5.
- Let s be the maximum number of categories shown to a worker who is judging relevance of categories to an item, default = 7.
- Let q be the minimum size of a category, default = 2.
- Let p be the percentage of items two categories must have in common in order to be deemed similar, default = 75%.

By changing the values of these parameters, the designer can trade off cost and running time against taxonomy quality. Decreasing k or increasing t, c, and s will lower the cost of execution, presuming workers will still accept the task. Finding the optimal values for these parameters is the subject of future research.

3.2.4 Primitive Worker Tasks

Before describing Cascade's overall control flow, we first present the three types of HITs that are shown to workers. The order in which these tasks are generated depends on the characteristics of the input items and can be complex, but the primitives are individually

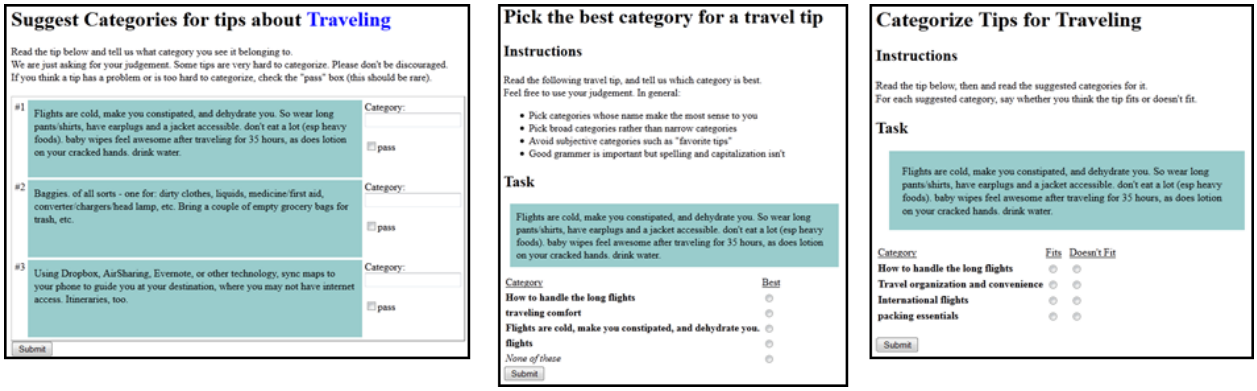


Figure 3.5: HIT Primitives: Generate, SelectBest, Categorize.

quite simple. Here we present them abstractly; Figure **??** displays concrete instances of these tasks.

- *Generate*: (t items) $\rightarrow$ t categories

  The Generate HIT presents a worker with t items and asks her to generate one suggested category for each item. The categories do not need to be distinct, but they often are. She may elect not to name a category for some items.

- *SelectBest*: (1 item, c categories) $\rightarrow$ 1 category

  The SelectBest HIT shows a worker a single item and c distinct category tags and asks her to pick the single best tag.

- *Categorize*: (1 item, s categories) $\rightarrow$ bit vector of size s

  The Categorize HIT presents a worker with a single item and s categories and asks the worker to vote whether the item fits each of the categories.

3.2.5 Algorithm Steps

Described at the highest level, the Cascade algorithm takes every item and solicits multiple suggested categories for it from different workers. A new set of workers then votes on the

best suggested category for each item. Cascade then asks workers to consider every item with all of these "best" categories and judge relevance. Cascade next uses this data to eliminate duplicate and empty categories and to nest related categories, creating a hierarchy. We first present an outline of the steps of Cascade and then describe the steps in full detail.

0. Select m items to be the *initial item-set*. All other items belong to the *subsequent item-set*.

1. Show each of the m items to k workers, who each *generate* a category suggestion for each item.

2. Show each of the m items and the category suggestions for it. Ask k workers to *select the best category* for each item by voting. Filter out suggested categories with insufficient votes.

3. For all items, for all best suggested categories, ask k workers to vote whether the category applies to the item or not. This *categorizes* the items.

4. Using the item-category membership information, run a fully-automated process, called *global structure inference*, to produce a taxonomy from the categories.

5. If there are items in the subsequent item-set, ask k workers to vote on whether the items fit into the categories already created.

6. Regenerate the taxonomy with the new item-category membership data.

7. If there are items that do not fit into any of the categories during step 5, create a new item-set from the uncategorized items. Repeat the algorithm (from step 1) on these uncategorized items.

The algorithm continues until all the items are categorized. Following are the implementation details for each step.

Step 1. Intentional Category Over-Generation

HIT primitives: *Generate* is called $\lceil m/t \rceil$ * k times.
Output: k suggested categories for each of the m items.

The first step of Cascade is to show each item to k=5 people, and have each of them suggest a category for it. We present items in groups of t=8. Although multiple items are displayed together, category suggestions are independent – workers do not have to name categories that apply to multiple items in that HIT. We display multiple items together so that workers get more context about the item-set as a whole. An additional benefit of showing multiple items together is that workers can easily skip items in the HIT. If a worker is unable to think of a category, it is better for her to skip it than to suggest something awkward, forced, or overly-specific. We allowed workers to skip at most 2 of every 8 items.

Step 2. Best Category Suggestion Vote

HIT primitives: *SelectBest* is called $m*k$ times.
Output: a set C of best suggested categories (of size ˜$1.5*m$)

In Step 1, k=5 individual workers attempted to categorize each item, resulting in up to five suggested categories for each item. If an item was skipped in Step 1, there will be fewer than k=5 suggested categories for it. If any of the suggested categories are exact-string duplicates, we remove the duplicate, also leaving us with fewer than k=5 suggested categories. In this step, we show each worker one item and $c \leq k$ of its remaining suggested categories. Workers vote for the one they think is best or select "None." Any suggested category that gets two or more votes is passed on to the next step. We call these the best suggested categories, and refer to the set as C.

Step 3. Adaptive Context Filtering

HIT primitives: *Categorize* is called $m * \lceil |C|/s \rceil * k + m*k$

Output: item-category membership decisions

Phase 1: We take the best suggested categories and partition them into groups of s=7 categories. In a *Categorize* HIT, we present each item with each category group to k=5 workers. Workers vote on whether each item fits each category or not. Any category that gets at least g=2 out of five votes goes to Phase 2.

Phase 2: In a *Categorize* HIT, we present each item with all the categories that had at least g=2 votes for fitting in that item. There will hopefully be fewer than s=7 categories. If there are more, we break the categories into group of $s = 7$. We get k=5 workers to vote, and any category that gets at least h=4 out of five votes officially fits in that category.

We call this 2-phase process *Adaptive Context Filtering*. We do Adaptive Context Filtering instead of straight categorization because we observed that workers vote differently depending on the group of categories we present to them. We observed that workers are often tempted to select at least one category that fits an item, even if none of them fit very well. Thus, we treat the categories found in Phase 1 as *potentially applicable categories*. In Phase 2, we vote on all the potentially applicable categories for an item together, as a group. When all the categories are at least potentially applicable to the item being shown, the context for selecting the categories is better and workers make better decisions.

Step 4. Global Structure Inference

HIT primitives: none

Output: a taxonomy with m items

At this stage, we have set of category labels generated by workers. We filtered out some of the category labels, and then we categorized all m items into all the remaining

Table 3.1: Asymptotic running time of Cascade with n=64, m=32 and other values at their stated default values.

	Running time	HITs	Cost
Step 1 : Intentional Category Over-Generation	$O(\lceil m/t \rceil * k)$	20	$3.20
Step 2: Best Category Suggestion Vote	$O(mk)$	160	$3.20
Step 3. Adaptive Context Filtering – Phase 1	$O(m * \lceil \lvert C \rvert / s \rceil * k)$	1100	$22
Step 3. Adaptive Context Filtering – Phase 2	$O(mk)$	160	$3.20
Step 5. Categorize Subsequent Item-Set	$O((n-m) * \lceil \lvert C \rvert / s \rceil * k)$	320	$6.40
Iteration 1 total		1760	$38
Iteration 2 total		1760	$38
Total:		3520	$76

category labels. Global structure inference uses this category membership data to organize the categories into a cohesive taxonomy. There are three steps:

1. *Remove insignificant categories.* Remove any category that has fewer than q=2 items.

2. *Remove duplicate categories.* For any two categories that share more than p =75% of their items, we remove the category with fewer items, breaking ties by random choice.

3. *Create nested categories.* For any category c_{sm} that shares more than p=75% of its items with another category c_{lg}, make c_{sm} a subcategory of c_{lg}.

The result of global structure inference is a taxonomy where all categories have at least two items, sibling categories represent distinct concepts, and subset categories are properly nested under their super category.

Step 5. Categorize Subsequent Item-Set

HIT primitives: *Categorize* is called $(n-m) * \lceil |C|/s \rceil * k$ times
Output: updated item-category membership decisions

Categorize the subsequent item-set on the existing categories.

Step 6. Update Taxonomy

HIT primitives: none
Output: a taxonomy with n items
Rerun Global Structure Inference on the item-category membership data from Step 5.

When to Recurse

The taxonomy is complete if it sufficiently explains all the items. There are two conditions for items that are not sufficiently explained:

1. Items completely uncategorized.

2. Items only in categories with 20 or more items.

If either of these conditions are met, we create a new item-set of the insufficiently explained items and rerun the algorithm from Step 1. As the algorithm reruns and generates new categories for these items, we apply the original items to the new categories as well (as in Step 5), and rerun global structure inference. This produces a taxonomy with new categories that explain the previously unexplained data.

It is obvious why we would want to rerun Cascade while there are items that are still uncategorized. However, it is less obvious that we need to rerun Cascade while there are items that are categorized, but only in large categories. We call items only in large categories "loosely categorized items." If a worker suggested a category that has 50% of the items in it, we want to make sure we find subcategories within that category to help users browse the items in it.

The total cost and running time of Cascade is dependent on worker output and how many iterations are needed. An example of the cost and running time for a dataset with two iterations (n=64 and m=32) is presented in Table **??**.

3.3 Experiments

To test the performance of Cascade we run the algorithm on three datasets obtained from Quora.com and present the taxonomies it produces.

3.3.1 Datasets

Quora.com is an online question-and-answer site. Often the questions are open-ended, such as "What are your best travel hacks?" Hundreds of people respond with valid tips and opinions. The responses are all valid, and this is the type of domain where a taxonomy would help users get a global picture of the data and navigate the responses. We picked 3

Table 3.2: Topics and size of item-set

Abbreviation	**Topic**	**# items**
editWriting	"What are some tips for editing your own writing?"	22
sideProjects	"How can I increase my productivity on my side projects at the end of the day when I?m tired from work?"	67
travel	"What are your best travel hacks?"	100

open-ended questions posted on Quora.com and created a taxonomy of items in the replies. The three datasets are summarized in Table **??**.

Often, a single response will contain multiple tips in bulleted lists, numbered lists, or paragraphs. We manually broke these responses into separate items. Previously, we have had the crowd do this breakdown. It is a straightforward step but not a part of the Cascade algorithm.

We randomized the order of the items to avoid any effects of our workers seeing items in the order they were generated.

3.3.2 Implementation

We implemented the primitive HITs in HTML and JavaScript, which served as ExternalQuestions on Mechanical Turk (MTurk). To dispatch HITs, we used TurKit [15]. Python scripts were used to process data between steps.

3.4 Results

editWriting was the smallest item-set with 22 items. In one iteration, Cascade produced a taxonomy with 15 categories, 8 of which were top-level (Figure **??**). Global structure inference correctly eliminated three redundant categories and eliminated insignificant singleton

categories, such as "reformat" and "write, delete, rewrite."

sideProjects is a mid-sized domain with 67 items. In the first iteration, 32 items were used to generate a taxonomy with 22 categories. Subsequent processing generated no new categories, and global structure inference created the parent-child relationships as shown in Figure **??**. *travel* was the largest item-set with 100 items. In the first iteration, 32 items were used to generate a taxonomy with only 7 categories. That taxonomy left 66 of the 68 items in the subsequent item-set insufficiently categorized. The next iteration of Cascade yielded a taxonomy with 51 items, fitting all the items (Figure **??**).

3.5 Evaluation

The goal of Cascade is to produce a taxonomy that provides a global understanding of the items. To evaluate that goal, we ask and answer three questions:

1. Are the category labels in the taxonomy as good as labels created by experts?

2. Does the taxonomy have an appropriate hierarchical structure?

3. Is the cost and running time of Cascade competitive with hiring experts?

3.5.1 Good Category Labels

Taxonomies are inherently subjective. Experts often disagree on the categories and level of granularity of the taxonomy. However, given a small pool of experts independently categorizing a dataset, one would expect some of the same categories to appear in multiple experts' taxonomies. In order to compare Cascade's categories to those of experts, we paid four information architects to produce taxonomies for our three datasets.

We performed the following comparison on the taxonomies. For each data set, we took the Cascade-produced taxonomy and the four expert taxonomies. We compared two things:

1. What fraction of the Cascade taxonomy categories are also named in at least one expert taxonomy?

Figure 3.6: Taxonomies created by Cascade.

Table 3.3: Category name quality comparison

	edit-Writing	**side-Projects**	**travel**	**Avg.**
% of Cascade categories shared by at least one expert	47%	50%	53%	50%
avg % of expert categories shared by at least one other expert	32%	70%	64%	55%

2. What fraction of expert categories are named in another expert taxonomy?

Table **??** contains the results of this comparison. For all three datasets, about 50% of Cascade's categories were also named by an expert. For example, in the editWriting dataset, four out of four experts named a category closely matching Cascade's category "working off an outline" (two experts named it "outlining.") The authors of this paper made the similarity judgments.

When comparing experts to each other, the average expert matching fraction was 32%, 70%, and 64% for the three datasets. This averages to 55% of categories matching another expert?s categories across these three hierarchies, compared to the 50% agreement between Cascade and the experts. Comparing these, we note that Cascade had 91% of the category agreement the experts did among themselves.

3.5.2 Mistakes in Hierarchical Structure

Cascade infers a global understanding of the data from the item membership of categories. Cascade removes categories that do not have enough items in them, removes categories that have a high item overlap, and creates a parent-child relationship for categories where one category has high item overlap with the other. These inferences are based on many

Table 3.4: Errors in hierarchical structure

	edit-Writing	**side-Projects**	**travel: iteration1**	**travel: iteration2**
# categories	15	18	7	51
Duplicate Categories	2	2	0	2
Missing Nesting	0	0	0	5
incorrect Nesting	0	3	1	3
Correct Nesting	5	3	1	23
total errors	2	5	1	10
Error rate	13%	27%	14%	20%

small judgments by potentially hundreds of different people. We want to know if all those judgments come together to form a coherent hierarchy. In particular, we are looking for three types of mistakes in the Cascade hierarchies:

1. Duplicate categories

2. Missing Parent-Child Relationships

3. Incorrect Parent-Child Relationships

To find the error rate in the hierarchical structure, we divide the number of errors by the number of categories in the taxonomy. The authors of the paper judged the errors. *editWriting* has the smallest error rate of 13% (Table **??**), with only 2 errors in 15 categories. Both were duplicate-categories errors. The categories "tips to edit better" and "how to edit better" should have been the same, but Cascade left them both in the taxonomy.

sideProjects had the highest error rate of 27%. This came from 3 incorrect parent-child relationships: "prioritizing" was the parent of "commitment," "prioritizing" was the also parent of "consistency," and "motivation" was the parent of "relaxation." In our judgment,

there is no clear reason that prioritizing should be a parent of commitment or consistency, or that motivation should be the parent of relaxation, and thus it is a mistake in the hierarchical structure of the taxonomy. These are errors produced by the automated global structure inference step. It nested "commitment" under "prioritizing" because more than 75% of the tips about commitment were also about prioritizing. Although these categories share many tips in common, they aren't semantically related: this is a danger of machine steps. Perhaps a solution would be to have humans check the resulting taxonomy for obvious errors.

Across the three datasets, the average error rate was 18.5%.

There were an impressive number of correct parent-child relationships, especially in the travel dataset, with 23 correct parent child relationship and 3 incorrect ones. Many air-travel and flight-related categories with complicated nesting are expressed with coherent hierarchical structure. For example, "air travel tips" is a parent of "flights," which is a parent of "flight layovers."

3.5.3 Time and Money

It is non-trivial to compare the costs associated with creating a taxonomy with Cascade versus experts. There is a cost-quality-time trade-off. For example, on MTurk, under-priced HITs will eventually get done, but will take a long time. The most basic comparison we provide is the actual costs and times in our run of Cascade and that of our recruited experts (Table **??**). Cascade took ˜6.5 times longer to complete the HITs, and was 1-3 times as expensive. However, the prices were set fairly arbitrarily. We paid our experts $25/hour as a set wage. We paid MTurk workers $0.05 per HIT. The average time to complete a HIT was 21.46 seconds. This equates to $8.39/hour, which is high for MTurk, where $3-$4/hour is more typical. Running the HITs at this standard marketplace rate would reduce the cost of Cascade by a factor of 2, making Cascade's cost competitive with the wage we offered experts.

The total time spent on all three datasets by the average expert was 6 hours 50 minutes, and the total time spent by MTurk workers was 43 hours 3 minutes. This is a factor of

Table 3.5: Time and cost comparison

	edit-Writing	**side-Projects**	**travel:**
Cascade Time	7 h 56 m	16 h 13 m	16 h 32 m
Avg expert time	1 h 23 m	2 h 36 m	2 h 5 m
Cascade Cost	$35.40	$109.45	$224.45
Average Expert Cost	$34.87	$65.13	$71.38

6.3 more time spent by MTurk workers. Seeing as the work done by workers is basically replicated k=5 times over, the time it would take for a single person to run Cascade on himself (k=1) would be competitive with the expert?s time.

More important than comparing total time spent on the algorithm is to think about the minimum amount of time that it would take to run the algorithm if sufficiently many people work in parallel, as is supported by Cascade. Each worker spends on average 21.3 seconds per HIT, and all the HITs in any step can be run in parallel. Thus, assuming Cascade is run in two iterations of 5 steps each, and the maximum time a worker spend on a task was 30 seconds, the entire time it would take to run Cascade would be five minutes.

3.5.4 Discussion

The Cascade algorithm is built on three simple HIT primitives: Generate, SelectBest and Categorize. These primitives are seen in other crowd workflows: Generate is very similar to what the ESP Game [**?**] and VizWiz [**?**] ask for. SelectBest is similar to the voting steps in the iterative improvement workflow used by TurKit [**?**] and Soylent [**?**] and Categorize is explored in Polarity [**?**] and is a common task on MTurk – it was the original reason Amazon created MTurk. Because Cascade uses common HIT types, workers do not have to spend the overhead of reading special instructions or learning a complex task.

Cascade is essentially bottom-up. We solicit many categories based on single items, we filter out bad categories and then use global structure inference to create a cohesive global

picture out of individual Categorize HITs. Global structure inference is the reason Cascade works. It is what combines small, independent contributions into a taxonomy and decides which items we need to rerun Cascade on, thus focusing work where it is needed most. Cascade allows items to go in multiple categories. This is what allows Cascade to perform global structure inference. For example, we know that "LAX security lines" is a child of "air travel" because all the tips in "LAX security lines" are also in "air travel." Additionally, we found that tips naturally fit multiple generated categories along different facets. For example, the tip "use kayak.com" is both about "saving money" and "air travel." We see no reason to make workers pick which category it fits better.

In Cascade, we compare data to abstractions, namely we compare items to a categories. We do not compare items to items, or categories to categories. We observed that workers were more comfortable with item-to-category comparisons. Comparing items to each other involves an assumption about the aspect of the item the worker is comparing. The item "use kayak.com" is similar to the item "free wifi at LAX" because it has to do with saving money and similar to the item "I hate Travelocity!" because it involves a website. It's hard to say which it is more similar to. Comparing categories to categories involves assuming a grounding for the abstraction. (Is the category "air travel" the same as "flights?") However, "free wifi at LAX" is clearly about "air travel."

3.6 Summary

This paper presents a crowd algorithm that generates a taxonomy over a set of independent data items, such as travel items, color blocks (Figure **??**), or images (Figure **??**). We show that using three HIT primitives – Generate, SelectBest, and Categorize – we can power an algorithm where workers do as little as 20 seconds of work. A crucial step in the algorithm is to use global structure inference to combine small, independent units of work into the final taxonomy. Compared to expert information architects, the taxonomies Cascade produced were competitive in quality and price. Since Cascade is parallelizable and uses small units of work, it can make use of large crowds of people and complete in minutes rather than hours.

Chapter 4

FRENZY: COLLABORATIVE DATA ORGANIZATION FOR CREATING CONFERENCE SESSIONS

Organizing conference sessions around themes improves the experience for attendees. However, the session creation process can be difficult and time-consuming due to the amount of expertise and effort required to consider alternative paper groupings. We present a collaborative web application called Frenzy to draw on the efforts and knowledge of an entire program committee. Frenzy comprises (a) interfaces to support large numbers of experts working collectively to create sessions, and (b) a two-stage process that decomposes the session-creation problem into meta-data elicitation and global constraint satisfaction. Meta-data elicitation involves a large group of experts working simultaneously, while global constraint satisfaction involves a smaller group that uses the meta-data to form sessions.

We evaluated Frenzy with 48 people during a deployment at the CSCW 2014 program committee meeting. The session making process was much faster than the traditional process, taking 88 minutes instead of a full day. We found that meta-data elicitation was useful for session creation. Moreover, the sessions created by Frenzy were the basis of the CSCW 2014 schedule.

4.1 Introduction

When planning an academic conference, organizers group papers into thematic sessions so that attendees can see related talks in the same time-block. However, constructing and arranging conference sessions can be a challenge, especially for a small group of organizers. First, organizers often need to consider a large number of accepted papers from multiple sub-disciplines. Knowledge about these papers is typically distributed within the community.

Second, accepted papers can be grouped in multiple ways (e.g., by topic, by problem domain, by method of study), so maintaining a global outlook of how and why papers fit together is often non-trivial. Third, sessions must satisfy two hard constraints: each paper must be assigned to exactly one session and every session should be the same length. This implies that even coherent paper groupings may be infeasible if they contain too many or too few papers, and that clever alternative groupings may be required to avoid stray papers or incomplete sessions. Since creating a session affects what other sessions can be created, the process of coming up with coherent sessions that satisfy global scheduling constraints requires effective coordination.

To manage this process, many conference organizing committees use printouts of the accepted paper abstracts. Through informal observations and interviews with organizers of two large conferences, we learned that a small group of dedicated organizers typically spend a day or two in person creating sessions by printing abstracts on cards, then tangibly arranging cards in piles, and trading cards between piles, until the piles roughly form session-sized groups. After all this, the information on the cards is manually entered into a computer.

This time-consuming process has several shortcomings. First, the one-to-one correspondence of a paper to a physical card limits the number of people that can work on grouping a paper at one time. Second, connections between papers are often made organically as an organizer walks around the room to collect related cards. This can lead to sessions with odd papers mixed in. Third, the time constraints and difficulty of navigating through piles of cards makes it difficult to consider parallel alternatives for group-ing papers. Organizers often feel "locked in" with these initially created sessions, because any modification requires them to add to or break apart another session. Further, to account for stray papers, organizers often leave the meeting with thematic sessions that contain too many or too few papers, leaving additional work for refining the schedule.

We introduce Frenzy, an alternative approach for creating sessions that draws on the distributed knowledge of the entire program committee. First, Frenzy breaks the task of session making into two sub-problems: meta-data elicitation and global constraint satisfac-

tion. During meta-data elicitation user can see all the accepted papers, search them by text, and add two types of meta-data: they can add suggested categories for a paper and they can indicate that they like a suggestion category and think it has "high session making potential" by clicking a "+1" button. This stage allows us to use many experts in parallel to contribute their intuitions about likely sessions. This stage can be done during the breaks of the program committee meeting, or after the wrap-up meeting. The next stage is global constraint satisfaction which uses a smaller group of volunteers who are co-located and can communicate easily with one another to use the suggested categories and "+1" votes to solve the global constraint of assigning all papers to exactly one session that has 3 or 4 papers.

To enable both stages we introduce a web application called Frenzy that facilitates parallel collaboration among large and small crowds. Frenzy uses some familiar concepts from social media such as Twitters' "feed" and Facebook's "tagging" and Google Plus' "+1" button to help view and add meta-data to papers. It also has standard features such as search and autocomplete. Frenzy allows parallel collaboration by providing each user with their own view of the data which they can search without affecting other users, but which propagates all their meta-data to other users immediately to eliminate redundant meta-data and give an active sense of collaboration.

We evaluate Frenzy by deploying the tool to the CSCW 2014 Program Committee, at which 48 committee members contributed to Frenzy. The session-creation meeting took 88 minutes compared to the traditional process, which often takes all day.

This system makes the following contributions:

1. We introduce Frenzy, a collaborative web interface for constrained data organization that uses goal completion with actionable feedback to alert users of what needs work, while providing them freedom to choose their task.

2. We address the challenge of using groups at different scales to collectively satisfy constraints and achieve a cohesive global structure by decomposing the problem of creating sessions into two sub-problems: meta-data elicitation and constraint satisfaction.

3. We evaluate Frenzy by deploying it to a conference program committee over two days. We show that providing actionable feedback allowed users to pursue their own strategies for completing the goals. We show that the meta-data elicited was useful for session creation. Moreover, the meta-data collected from Frenzy was more useful that the legacy paper categories used by that conference.

The paper proceeds as follows. We first describe related work in crowdsourcing and groupware. Next, we discuss the design motivation for Frenzy and describe the system and its implementation. We chronicle the deployment of Frenzy to the CSCW 2014 Program Committee (PC) with data analysis and interviews. The sessions created by Frenzy were the basis of the CSCW 2014 schedule.

4.2 System Description

The Frenzy interface (Figure **??**) consists of four sections: the query bar, results feed, results hyperbar, and goals with actionable feedback.

The query bar is a good place for users to start exploring Frenzy by performing a text search over papers. A paper matches the query text if the paper's title, author list, author affiliations, or abstract contains that text. Text search helps users narrow the list of papers by broad terms such as "Crowdsourcing", and also helps retrieve particular papers by an author or from a keyword in the title. Users can also see all the papers by clicking the "Show all papers" button.

The results of the query are displayed in the results feed which shows a vertical list of data cards (Figure **??**) for each of the papers returned by the query. A data card displays the papers information (title, authors, and abstract) on the left side and the user-generated meta-data box on the right. The meta-data box contains a list of categories suggested for the paper, as well as a count of how many people have added a "+1" vote to the category, indicating that they think they category has high session potential. If the users are in the constraint satisfaction stage of Frenzy, there is also a text box to enter a session name. Users

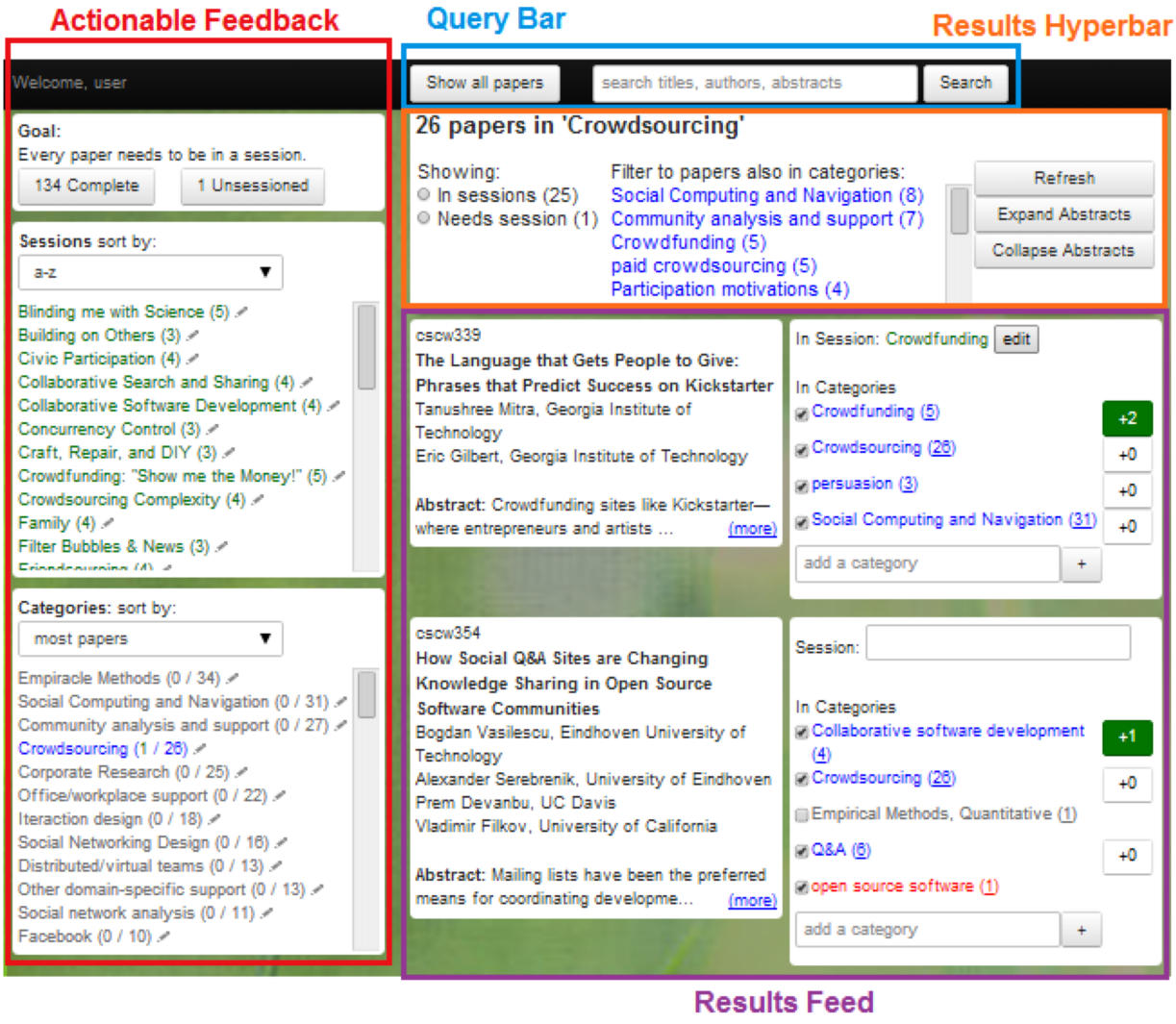


Figure 4.1: Frenzy interface, highlighting 4 sections: actionable feedback, query bar, results hyperbar, and results feed.

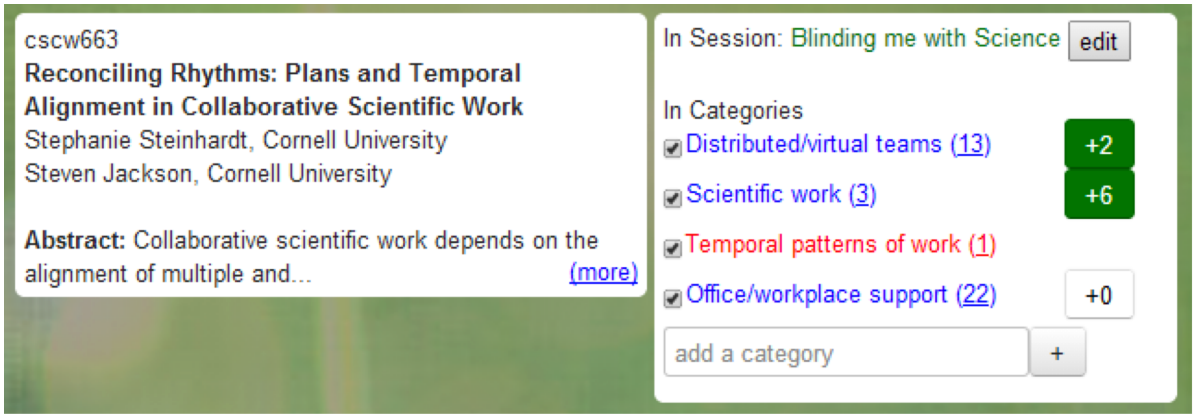


Figure 4.2: The data card for paper cscw663. The left side is paper details and the right side is the meta-data box

may add their own meta-data by adding a category, adding their own "+1" vote, adding or editing the session name (if applicable) or removing a category from the paper. We call this area the results feed because it draws some similarity to the social media concept of a feed of updating information. The interface draws visual connections to the Twitter interface, which is familiar to most of our users.

When the results feed is updated by a query, Frenzy also update the results hyperbar. The results hyperbar displays feedback about the query that was performed: how many results are returned, and a statement of the query. It also returns additional filters which the user can apply to their query. For example, if the user searches for "Crowdsourcing" the results hyperbar will say "26 papers in 'Crowdsourcing' " and will list up to 5 categories the user can filter by, such as 'paid crowdsourcing.' If the user clicks these additional filter categories, the results will update to have "5 Results for papers in 'Crowdsourcing' and 'paid crowdsourcing.' " These additional filters make it easy to drill down into large categories.

The left panel of Frenzy displays the actionable feedback. In both stages of Frenzy (meta-data elicitation and constraint satisfaction) there are two goals for users to achieve as a group. For example, in the meta-data elicitation stage one of the goals is: "Every category must have at least two papers in it (No singleton categories.)" Instead of instructing users

how to achieve this goal, Frenzy provides two types of feedback on progress towards that goal that users can easily take action on (hence the name actionable feed-back.) One type of actionable feedback for this goal is the list of categories, with the number of papers in that category in parenthesis. Any categories with only one paper are displayed in red text, indicating a problem. Users can then click on that category to see what paper is in it, and either remove the paper from it (thus deleting the category), or add more papers to the category. An additional type of actionable feedback is the number of papers that meet the goal and the number that do not. By clicking the number of papers that do not meet the goal, users generate a query that returns papers that need work. This provides an easy and direct way to find papers in need of contribution.

Frenzy is a web app implemented in node.js and Bootstrap. Each user logs in and then sees the Frenzy interface. When users query the system, their query is private to them. However, all meta-data generated is propagated to other users within 5 seconds. If a user adds a category to the paper, that category will appear in other users' data cards, and in the actionable feedback pane, the number of papers in that category will update, as well as the number of completed goals, if applicable.

To make Frenzy a flexible microtask platform, our design has three goals:

1. Enable lightweight contributions.

2. Allow users to access all the data and tasks.

3. Promote completion of goals with actionable feedback tailored to user groups and stages of problem solving.

We next describe how the Frenzy interface supports these design goals.

4.2.1 Design Goal 1: Enable Lightweight Contributions

In order to encourage as much participation as possible even in only short periods of free time, all contributions to Frenzy are small tasks that a user can complete in a under a minute.

Users can choose their contributions and move easily between tasks. This allows users to make contribu-tions that best fit their availability and expertise.

All user contributions are made using the meta-data boxes associated with each meta-data card. There are four ways a user can make contributions on each meta-data card:

Add a category

If a user can think of a new category the paper fits into, they are encouraged to add it. Autocomplete in the "add a category" textbox helps users reuse categories that are already in Frenzy. Additionally, categories with only one paper appear in red to indicate that they are single-tons.

Remove an existing category.

In order to remove a paper from a category, the user simply unchecks the category. Unchecked categories become less visually salient by turning grey, and their upvote button disappears. Category names remain visible and can be checked again to reassign a paper to a category.

Upvote a category.

We expect every paper to be assigned to multiple categories. Categories may represent different aspects of a paper, such as its topic (e.g., education, question answering, games, health) or contribution type (e.g., study, system) Categories may also vary in their levels of granularity or quality. Up-voting a category indicates that the category has high potential of becoming a session. Typically this means the category is small (3-8 papers) rather than being overly broad such as "Empirical methods" which has over half the papers in it.

Create/update/delete session names.

Users can place a paper in a session by entering a session name in the meta-data box. Session assignments can be deleted or edited at will. To distinguish sessions from categories, Frenzy

only allows each paper to be placed in a single session at any given time. This feature is only turned on during the constraint satisfaction stage.

Giving users more control and freedom also comes with potential disadvantages. First, users have to navigate the set of tasks and take the time to figure out how they will contribute. Second, seeing current work could potentially bias the results of future work. While some of these issues are mitigated by the actionable feedback presented to users, they also represent some inherent tradeoffs of having no fixed microtask workflow.

4.2.2 Design Goal 2: Give Users Access to All Data and Tasks

Lightweight contributions are convenient, but are only useful if users can find a place to contribute. Fixed workflows present users with a designated place to work, but we rejected the idea of using workflows in Frenzy because we wanted users to make contributions based on their expertise. Nobody knows a user's expertise as well as the user does, thus our solution to enabling contributions was to offer access to all papers and tasks through domain-specific search features. Because papers are attached to tasks to be done on those papers, users can search for papers that fit their expertise as a way for searching for tasks to be done that match their expertise. Frenzy supports text search over the titles, authors, affiliations and abstracts of the papers. Frenzy enables category-based searches in the actionable feedback panel, and the results hyperbar allows search results to be further filtered by relevant categories.

A benefit of this search-based solution to giving access to tasks is that it supports the existing discussion-based social process rather than replacing it. Users can discuss papers and then search for them, see their details and add meta-data that may results from that conversation. Search provides the user control and freedom [**?**] that communities want to feel in their collaborative efforts.

4.2.3 Design Goal 3: Promote Completion of Goals with Actionable Feedback

Showing hundreds of items gives users the control and freedom that they want, but can also be overwhelming. Frenzy provides actionable feedback to allow users to quickly find places

to work that need their attention and which they are knowledgeable about.

In order to effectively coordinate work between a large group of users providing meta-data and a small group of users making sessions, we associate with each sub-problem its own set of goals, based on which to present feedback.

In the case of meta-data elicitation, we set two goals:

MDE1. Every category must have at least two papers in it (No singleton categories)

MDE2. Every paper needs to be in a least one category with +1 for session-making potential

In the case of session constraint satisfaction to assign all papers to session, we set these two goals:

SCS1. Every paper needs to be in a session

SCS2. Every session must have more than 2 papers

Associated with every goal is visible feedback. For example, the MDE2 goal is displayed at the top of the actionable feedback panel with two buttons, one saying "100 Items Complete" and the other saying "35 items need work." This feedback is actionable because the user can click on the button and filter the results to only the items that need work, or that are already completed. For the MDE1 goal, Frenzy highlights any singleton categories in red. When the user clicks on the red category name and sees which paper it contains, they can either remove the singleton category from it (which deletes the category from the system because it then will have no papers in it), or find other papers for the category.

It is important to note that Frenzy does not assign tasks to users. The affordance in the actionable feedback is a shortcut for a search that nudges users towards finding a subset of the data to attend to. Once users find a place to work, there are many strategies for meeting the goals. A user may find that they need to add categories to a particular item, or remove them from another, or merge two categories, or split large categories into multiple smaller categories. Results from the psychology literature indicate that setting goals is an effective management style that gives people freedom in how they choose to achieve a goal [**?**].

4.3 Deployment

We partnered with the chairs of the CSCW 2014 Program Committee (PC) and deployed Frenzy for the initial session creation process. The traditional process involves 10-15 PC members meeting face-to-face after all the paper decisions have been made with the accepted papers' information printed on cards that they organize into piles which then become sessions. This process has two drawbacks:

1. it can only involves a limited number of people, and therefore fails to leverage the expertise of all the members of the communities, and

2. The amount of exploration is limited by that fact that there is only one copy of each paper that must be assigned to exactly one pile at any particular time.

This process tends to take the better part of a day.

Traditionally, session making can only start after all the paper decisions are final because the problem has constraints that do not make sense to solve until the data is complete. In contrast, Frenzy breaks down the problem into two stages: meta-data elicitation (referred to hereafter as MDE Frenzy) and session constraint satisfaction where the session-making is finalized (referred to hereafter as SCS Frenzy). MDE Frenzy and SCS Frenzy use the same Frenzy platform, but with minor adjustments to the actionable feedback portion of the user interface. In MDE Frenzy, the goals are MDE1 and MDE2 (see previous section) and the actionable feedback focuses on categories. In SCS Frenzy, the goals are SCS1 and SCS2 and there is actionable feed-back for both categories and sessions. All the meta-data gathered in MDE Frenzy appears in SCS Frenzy.

The CSCW PC received over 500 submissions. Of those, approximately 100 were fast-tracked for likely acceptance and approximately 100 were slated for discussion. We loaded the approximately100 fast-tracked papers into Frenzy before the meeting started, and as the committee made decisions about individual papers, they were manually added to the Frenzy interface. This way, Frenzy always contained a current view of the accepted papers.

CSCW has a set of 32 legacy categories for papers such as "Entertainment/games" and "Social Network Site Design and Use." The authors of the paper selected multiple of these categories that apply to their paper. We imported this pre-existing meta-data into Frenzy as categories. As a result, all papers had at least two categories at the start of Frenzy.

4.3.1 Meta-Data Elicitation (MDE) Frenzy

The PC meeting had 63 attendees, 43 of whom participate in meta-data elicitation for Frenzy. At PC meetings, there are several times when certain members must step out of the room and into the hallway due to conflicts with the authors of the paper being discussed. We used this free time by setting up Frenzy on a computer with a large monitor in the hallway. Thus, PC members could browse the accepted papers and enter meta-data during free time. Since PC members tend not to take their laptops with them into the hallway, this was the only computer available, and often multiple people gathered around the screen and discussed the papers and meta-data together. PC members also used the interface during breaks from the meeting and from their own laptops inside the meeting. After all the paper decisions were complete, the PC members engaged in a 5-minute session dedicated to entering meta-data in Frenzy.

4.3.2 Session Constraint Satisfaction (SCS) Frenzy

After the PC meeting ended, nine volunteers including the PC chairs moved to a smaller location to create a preliminary set of sessions for the conference. Sessions needed to have between 3-5 papers in them, ideally four. The number did not need to be exactly four since sessions were likely to change when refining the schedule, e.g., if presenters have conflicts. The goal was to have initial sessions to work from, and the actionable feedback encouraged sessions to have at least 3 papers.

The group started with the meta-data collected from the all the PC members, including 330 category entries made and 236 category upvotes.

4.4 Data Analysis

During the two days of Frenzy deployment, we logged all user actions: sign-ins, sign-outs, queries, and data entries. We observed people using the system and conducted inter-views with the SCS Frenzy participants. We now analyze this data to evaluate Frenzy. We show that the overall goal of session making was achieved in record time and success-fully incorporated the view of a large group of experts. We show that the design of breaking the problem into two sub-problems was effective by showing that the meta-data collected in MDE Frenzy was used extensively in SCS Frenzy. We show that goals with actionable feedback in both MDE Frenzy and SCS Frenzy provided the users the control and freedom to define their own strategies for successfully achieving their goals.

4.4.1 Achieving Overall Goals

Frenzy saw substantial usage over the 2-day deployment. A total of 48 participants contributed over 10.28 hours of usage. In that time, 2,365 queries were issued and 1,088 meta-data contributions were made. Over 250 contributions were made during the 5-minute period where all PC members were asked to spend 5-minutes simultaneously using Frenzy. This demonstrates the capacity of Frenzy to allow simultaneous contributions from a large group of users.

During SCS Frenzy usage, the 9 volunteers to complete the sessions achieved the goal of assigning every paper to a session and having every session have 3-5 papers in 88 minutes.

4.4.2 Breaking Session-Making into Two Subproblems

Frenzy breaks session-making into 2 sub–problems: MDE Frenzy and SCS Frenzy. We evaluate this design by testing whether the meta-data from MDE Frenzy was useful for making sessions in SCS Frenzy. The usefulness of category meta-data was tested by looking at how many of the papers ended up in a session that matched one of the categories give for it. The usefulness of upvote meta-data was tested using a logistic regression to model

the effect of upvoted categories on session creation. We defined a category as "matching" a session if one or more important keywords were shared between the category name and the session name.

How useful was category meta-data?

In total, 93 of 135 papers (68.9%) had a category that matched a session. The categories on those 93 papers could have come from two sources: the 32 predefined categories assigned by the PC or the 63 non-singleton categories con-tributed by users of MDE Frenzy. Although both are helpful in making sessions, only categories contributed by users of MDE Frenzy can be claimed as a benefit of asking users to add meta-data. Of the 93 papers with matching sessions, 40 of the matches came from predefined categories and 53 came from user contributions. MDE Frenzy more than doubled the number of useful categories.

How useful was upvote meta-data?

MDE Frenzy generated 99 non-singleton categories. After constraint satisfaction, there were 34 sessions, of which 25 matched categories. We want to know if the +1 voting for categories helped to determine which of the categories would be turned into sessions. To test if +1 voting provides a useful signal that a category will match a session, we run a logistic regression predicting the probability that a category will match a session (*prMatchesSession*). The dependent variable is an indicator of whether that category was the most +1 upvoted category for at least one paper (*wasMostUpvoted*).

Table **??** shows the coefficient estimates. A chi-squared test shows coefficent b is statistically significant ($p = 0.0082$). The interpretation of this logistic regression model is that for a category where *wasMostUpvoted* = 0, the predicted probability that it will match a session is 0.12. In contrast, for a category that was the most upvoted category for at least one paper (*wasMostUpvoted*=1), the predicted probability is 0.42. Thus, if a category is ever the most upvoted category for a paper, it has high session-making potential, which confirms the usefulness of upvote meta-data to session-making.

Table 4.1: A logistic regression predicting whether the probability a category matches a session is dependent on whether that category was the most upvoted category at least once.

$$prMatchesSession \sim a + b * wasMostUpvoted$$

Coefficient	Estimate	Std. Error	Pr(>|z|)
a	-2.0794	0.4330	1.57e-06
b	1.7658	0.5278	0.000821

4.4.3 Goals and Actionable Feedback in MDE Frenzy

The main mechanism that drives workers to make progress in Frenzy is having goals with actionable feedback. This helps users find a subset of the data on which to work in order to achieve the goals. Because we do not know what actual items are best to give to each worker and we do not know what tasks to give them (adding labels vs. removing labels vs. placing items into sessions), we provide an interface that grants users control and freedom (Design Goal 2) that allows them to find their own strategies for contributing towards the goal (Design Goal 3). In our deployment of MDE Frenzy we found 3 strategies for contribution:

Browse and Edit

Users often came to Frenzy because they were curious to browse the list of accepted papers and to see how the conference was shaping up. As they browsed, the meta-data box was clearly visible, and often they made a contribution. For example, by browsing the system, a PC member who was not an expert in education added the category "MOOCs." Another user noticed several papers about teens, adolescents or children and added the category "youth" based on several key word searches for "children," "teens," and "adolescents." The categories "email," "facebook" and "twitter" were added by users who noticed the terms pop up, searched for them to see if they were themes in the program, and then added the categories.

Check for Patterns within One's Area of Expertise

The papers already had good category labels given by the authors from a checklist provided by the conference organizers. Thus workers could enter Frenzy, select their subfield of ex-pertise and look over the existing meta-data. For ex-ample, a crowdsourcing expert selected the "Crowdsourcing" category, which had 26 papers. He looked over the list and found that a few of them had something in common. He added the category "Crowdfunding" to five of the items. These five papers were ultimately grouped together in a session called "Crowdfunding: Show me the money!"

Similarly, a social media expert searched for "Social Networking Site Design and Use" which had 16 papers and created the category "Politics/Social Media" which had 4 items. These four items were ultimately grouped in session called "Social Media & Politics."

Direct contribution

As PC members were actively discussing papers in the meeting, they would occasionally see connections or pat-terns during the meeting then want to come out to the hall-way to enter them into the system. For example, during the PC meeting discussions, one PC member realized two papers were accepted about social media and depression ("Social Structure and Depression in TrevorSpace" and "Characterizing and Predicting Postpartum Depression from Facebook Data"). At the next opportunity, they used Frenzy to do a text search for "depression" and added "Depression" as a category to both papers.

4.4.4 Goals and Actionable Feedback in SCS Frenzy

As shown earlier, the session making goals were completed in record time and SCS Frenzy made heavy use of meta-data collected in MDE Frenzy. Figure **??** shows data entry activity averaged over 1-minute intervals. Types of data entry and color codes and stacked. From this graph and the interviews conducted with the users, we identify four distinct stages of the session making process: additional data-entry, removing clutter, session making, and lastly

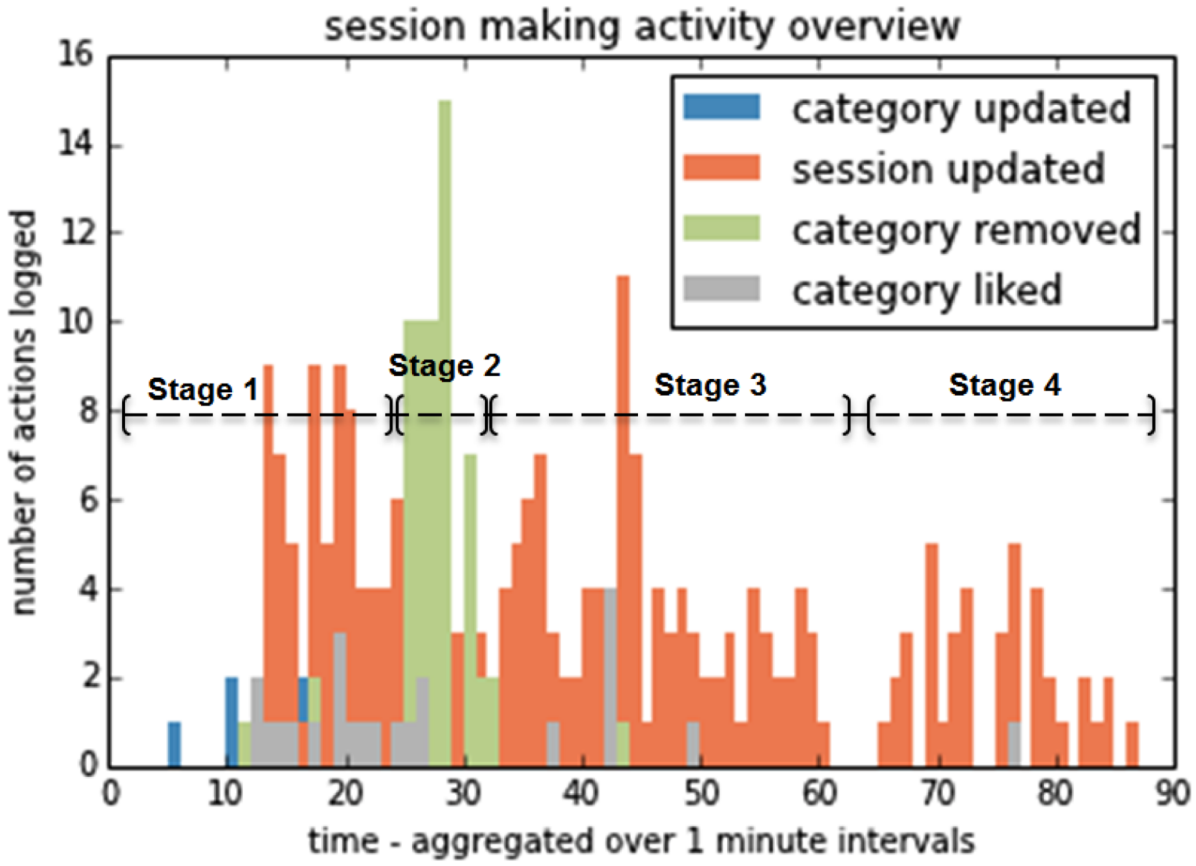


Figure 4.3: Data entry activity during SCS Frenzy. Data entries are broken into 4 types and stacked on the graph: category updates, session updates, category removal, and category upvotes.

session negotiation.

Stage 1: Additional data entry and low hanging fruit

Figure 3 shows that during the first 28 minutes, more categories were added and more upvotes were contributed. In addition, some of the more obvious sessions were created, such as turning the 4 papers in the category "Crowdsourcing" into a session called "Crowdsourcing: Show me the money!" We call such sessions made directly from categories low hanging fruit. During this period, volunteers largely worked alone without communicating to one another. One volunteer stated:

> *I started with stuff I knew... I have a pretty good sense if a paper belongs there or not.*

Stage 2: Removing Clutter

Figure 3 shows that from minutes 24-32, many categories were removed. A large number of the more obvious sessions had been created but now the low hanging fruit was gone. The volunteers pointed out a number of categories that were created for fun such as "Pacific Northwest Pride" and "What's Niki doing now?" They felt these were cluttering the data and making it hard to find meaningful meta-data, and so they had them removed. One volunteer commented:

> *I tried to remove some of the joke labels because they were starting to clutter up the user interface.*

Stage 3: Session making with discussion

Figure 3 shows that from minutes 32 to 62, more sessions were created. In contrast to the beginning of the process where volunteers with taking low hanging fruit and not communicating, there was lots of communication both to coordinate efforts and to get reassurance from others. Our volunteers said:

> *A couple of times I would try to make a session, but confirm verbally I was making that session [so] that they didn't have to do it.*

> *People would be working on their stuff, then somebody would speak up and say. Where could this paper possibly go?*

Stage 4: Session Negotiation

Figure 3 shows that from minutes 60 to 88, category addition slowed with many gaps in time where no sessions were updated. During this period, there was a lot of discussion about how to resolve the most difficult remaining constraints. Discussion dominated data entry, and the discussion resembled a negotiation with volunteers making proposals of switches and coming up with creative ways of reconfiguring existing sessions. For example, they took the 7 papers in "Q and A", found an additional paper on social networks and then split "Q and A" into two categories, "Q and A" and "Friendsourcing" which focused on using Face-book and Twitter friends for information needs. This creative problem solving is characteristic of the session negotiation stage.

User feedback

Our interviews revealed a consensus that Frenzy was an improvement over past experience with paper-based conference session-making:

> *It was definitely much less painful to create those initial sessions, [but that] it was more fun to do it with a social tool... it was nice to do it in a group, to have a few other people in the room... especially when we ran into tricky ones, helpful to talk it over.*

There was also consensus that face-to-face aspects of the collaboration during SCS Frenzy were still vitally important to the process:

I wouldn't want to do this remotely, having everyone there was important, awareness of what other people were doing, touching certain papers, or thinking about sessions.

In particular, the social interactions in the room were essential for resolving some of the session making problems: Sometimes people would go in different directions and make simultaneous edits. We had to stop and negotiate a little.

4.5 Discussion

While Frenzy is a special purpose tool for creating conference sessions, it solves a more general problem: collaborative data organization. There are three techniques Frenzy uses that are critical to enabling the collaborative process within the community.

4.5.1 Technique 1: Flare and Focus

As discussed in the related work section, flare and focus is an approach to ideation introduced in the design literature. Frenzy's two-stage process borrows from this idea: meta-data elicitation, wherein users generate categories and con-tribute +1 votes is similar to the flare stage, and constraint satisfaction, where the main focus is eliminating options and focusing on the best of the generated options. The flare stage is what allows Frenzy to use a large crowd of collaborators in parallel without worrying about how the constraints will be solved and the focus stage is where the constraints are solved.

One important detail of Frenzy's flare and focus process is that there is such a thing as too much flare. Between the flare stage and the focus stage, it is necessary to get rid of clutter created in the flare stage — categories clearly too large or too small, joke categories. Eliminating clutter is crucial to a successful focus phase.

4.5.2 Technique 2: +1 Voting

Voting is a common crowdsourcing technique. +1 Voting is slightly different from true voting because users do not necessarily have all the options set before them when they cast their +1 vote. +1 voting is an analog of voting which is not limited by the synchronization barrier: you do not need to control the workflow of microtasks to ensure that voting is only done after all the options are listed.

A subtle point about +1 voting is that if you want to include pre-existing categories, you need to have a mechanism like +1 voting in order to set reasonable goals for the users. Without pre-existing categories, a simple goal such as "every paper must have at least one category" will suffice. However, if there are pre-existing categories, that simple goal may already be satisfied. Changing the goal to say "every paper must have at least one category with a +1 vote" the means the users will look at every paper, even if it has already been categorized.

4.5.3 Technique 3: Social Interaction

Frenzy uses design patterns from crowdsourcing such as tagging and voting microtasks. However, microtasks alone, even in a platform that affords a great deal of user control and freedom, are not enough to solve the problem of session making. Face-to-face communication is crucial for solving many small but important issues: resolving conflicts, coming up with new terms that will be acceptable to the community, and arriving at a consensus that the process is complete and the results, while not perfect, are satisfactory. One of the major lessons Grudin [**?**] derives from the failures of groupware systems is that social processes have subtle inner workings that we do not always understand, are difficult for computers to have access to and reason about, but are essential to the process. Thus we should not seek to replace the existing social process with a rigid workflow, but to add lightweight microtasks on top of the current process.

Frenzy is a hybrid approach which can take advantage of the lightweight and efficiently

parallelized contributions of microtasks and still incorporate the inspiration and social cues that drive consensus which transpires from face-to-face communication. Our general observation is that microtasks do not have to be distributed to strangers in rigid workflows, they can be made social and can enhance face-to-face interaction rather than aim to replace it.

4.6 Summary

In this paper we present Frenzy, a tool for collaborative data organization applied to the complex task of conference session making. Frenzy enables groups of experts to simultaneously contribute by breaking the problem into two sub-problems: meta-data elicitation and session constraint satisfaction. Frenzy gives users control and freedom in identifying their own strategies for accomplishing set goals. Actionable feedback promotes meeting goals and steers users toward useful work.

In our deployment of Frenzy at the CSCW 2014 PC meeting, we evaluated the actionable feedback features by identifying three strategies used to satisfy the goals. We showed the benefit of breaking the problem into two sub-problems by showing that the meta-data collected in MDE Frenzy helped form more sessions that the categories created by the PC before the PC meeting. Moreover, the sessions created by Frenzy were the basis of the CSCW 2014 schedule.

Chapter 5

HUMORTOOLS: AN ADAPTIVE WORKFLOW FOR A CREATIVE TASK

5.1 Introduction

Humor is a highly valued human skill. It is a sign of creativity and intelligence that enriches many domains: entertainment, advertising, social bonding, education and journalism (e.g. *The Daily Show*). Its engaging and surprising nature make it a compelling and memorable form of communication. As an intellectual challenge, humor has been studied by great Western thinkers – Plato, Kant, Freud [**?**] – and by modern philosophers, such as Daniel Dennett [**?**], and linguists [**?**, **?**], all of whom have found humor difficult to describe.

Humor generation presents a grand challenge for artificial intelligence because it requires language abilities, world knowledge, and understanding of how people process information that computers do not yet have. Furthermore, many people find it challenging to create humor on demand. For those who can, it is a process that is almost entirely internalized and difficult to explain. This leads us to think that it requires the magical "black box" of creativity.

Generating humor in a systematic, externalized process requires decomposing it into simple steps. There is no obvious decomposition of humor so to approach the problem we take inspiration from philosophers' and linguists' descriptive models of humor. The descriptive models are often insightful. For example, the benign violation theory of humor postulates that humor violates our expectations [**?**]. This model feels intuitively true and insightful. Unfortunately, it is too abstract to apply directly. To generate humor, we need concrete mechanisms that embody abstract models. With the descriptive models in mind, we performed an analysis of 330 jokes from *The Onion* looking for concrete mechanisms that

fit or refined the descriptive models. Although these concrete mechanisms are simple, they still require people's knowledge and language abilities to apply. Thus we propose a set of microtasks that can be used in a systematic and decomposed process for generating humor.

Traditional crowdsourcing uses microtasks in static workflows, such as *iterative improvement* or *find-fix-verify* [**?**, **?**] to combine microtasks into a useful output. However, these simple patterns are insufficient for the difficult problem of humor creation. Instead, we must apply microtasks in an adaptive workflow that allows users more freedom to selectively choose microtasks that fit the current context of the problem.

This paper presents HumorTools, a systematic process for generating humor through microtasks combined in an adaptive workflow. HumorTools models the humor generation process as a constraint satisfaction problem that is met by searching a conceptual space. The constraint is to violate an expectation, and the conceptual space to search is defined by a 20 microtasks such as coming up with associations for words, writing down expectations in a given situation, and imagining the expectations and beliefs from unexpected points of view. The adaptive workflow asks workers to iteratively apply microtasks in a brainstorming-like fashion, until they can meet the constraint of violating an expectation. Not all chains of microtasks will lead to an expectation violation, so users may need to backtrack and follow a different chain of associations.

This paper makes the following contributions:

- a survey of humor research and humor writing processes described by professional comedians
- twenty novel microtasks, distilled from a large-scale analysis of professional news satire, that are useful for humor generation
- an adaptive workflow that poses humor generation as a constraint that can be satisfied by searching a conceptual space
- the HumorTools implementation of our adaptive workflow and microtasks, and an

evaluation of its performance on 20 people, finding that the 75% of users were able to generate humor in this externalized way, users reported that the workflow make their process methodical and enable them to make a wider variety of jokes. Three of their jokes were rated as funnier than *The Onion*.

5.2 Survey of Humor Literature

To inform our design of a microtask-based, humor-creation workflow, we wanted to derive vocabulary, theories, and process ideas from experts in humor.

5.2.1 Theories of Humor

Humor has been studied for thousands of years. It has been explored by Ancient Philosophers such as Plato and Aristotle, modern philosophers like Kant and Schopenhauer, and is an active field of study in linguistics and psychology. But by and large, these thinkers have attempted to define humor and to understand *why* things are funny, not to mechanize the humor-generation process. The three major categories of humor theories found in the literature are superiority, release, and incongruity.

- **Superiority theories** claim that humor is the result of feeling superior to somebody. Plato and Hobbes describe all humor as having a basis in insult, when the listener understands the insult, they can feel superior to the people being insulted.

- **Release theories** claim that humor is release of tension built up from the suppression of discussion in society. Freud is the the champion of these theories noting that taboo subjects are often the target of jokes particularly because of the emotional release they allow.

- **Incongruity theories** claim that humor comes from the realization of an informational anomaly. The idea dates back to Aristotle but has been the focus of most modern humor theories. Benign Violation Theory [**?**] claims that humor comes from detecting

> an incongruity between expectations and reality, as long as the reality is not overly offensive. Kant, and later Schopenhauer, claimed humor was the result of perceiving a metaphor and thus seeing something in a new way that you would have previously thought unrelated.

Incongruity theories allow theorist to understand why jokes are funny by modeling the cognitive process of listeners. For example, according to Sul [**?**] (and others) an incongruity is not just an instance when expectations are violated, because there are many examples when expectations are violated that are not funny. To be funny, the incongruity must also *resolve* in the listeners mind. In Sul's cognitive model of humor appreciation, a joke has a setup in which the reader predicts an expected outcome. At the endif the listener predicts the outcome, then there is no surprise and thus no laughter. If there is surprise, then there is incongruity. If the listener is confused, then the incongruity is not resolved and it is not funny. However, if the listener can find a logic that makes the (unexpected) ending follow from the set up, then it is funny.

The Semantic Script Theory of Humor (SSTH), presented by Raskin in 1985 [**?**], is a widely accepted incongruity model of humor. According to SSTH, each joke can be interpreted according to two distinct, opposing scripts. One of those scripts is usually an expectation based on the set up, and the other is often an inference based on the punchline. An important aspect of this theory is that the incongruity is often in the subtext of what is said, and not the text itself.

5.2.2 Survey of Humor Advice from Experts

While many people assume that humor is a completely subconscious process that cannot be taught, numerous comedians have written books claiming that humor is learnable and that there is a conscious process for creating it. We selected five popular books by comedians, describing aspects of their humor creation process. We refer to each book by the authors' last name: Carter [**?**], Dean [**?**], Kaplan [**?**], Holloway [**?**], Vorhaus [**?**].

Structure: Setup and Punchline

Jokes have set-up and punchline — the set-up establishes expectations and the punchline violates those expectations. The setup should be *relatable*, these are easy to describe in brief and will lead listeners to make many assumptions. Punchlines violate expectations by saying something that fits with the setup, but is unexpected. This aligns well with many incongruity theories of humor. Additionally, three of the books argue that the core of a punchline is truth. In summary, a good punchline is insightful — it helps you see something new about a familiar situation or object.

Exploration

All the comedians agree that writing a joke requires exploration of a topic. Some explorations may lead to jokes, but many will fail, and you cannot know which ones will work in advance. This aligns well with the design literature on brainstorming and ideation. Holloway explicitly mentions mind maps, referring to them as "joke webs," as a way to explore a topic. Exploration can be used at many stages of the joke writing process. Carter emphasizes its use to determine a good premise for a joke, while Dean discussion the need to consider many alternative punchlines for a given set up.

Details

Jokes, like most good writing are richer when they contain details. Details make words more vivid to an audience, draw people more deeply into the work, and increases their emotional investment. Carter and Vorhaus both express a connection between details and truth. When a comedian strives for details, she has a better sense of the actual idea she is trying to portray. Carter advises that a premise for a joke should express a detailed picture of the emotion it is trying to portray.

Point of View

All of the comedians reinforced the importance of point of view in constructing jokes. Most people naturally tell stories or jokes from their own perspective. But the material becomes more immediate when it is told from the point of view of the person directly affected. Here is an example of a joke that has had its point of view transformed from neutral to immediate:

Before:

Q: How many Amish does it take to screw in a light bulb?

A: Two: one to screw in the light bulb, and one to wonder what it's for.

After:

Q: How many Amish does it take to screw in a light bulb?

A: What's a light bulb?

In the second version, where the listener is assumed to be the Amish person, the punchline is shorter, more immediate, and funnier.

Assumptions and Inference

Jokes are a story stripped down to its essentials. When writing a joke, the writer relies on the listener to make assumptions. Dean's example of this is the following: *"My wife just ran off with my best friend. Boy, do I miss him."* The assumption from the setup is that he misses his wife. The inference from the punchline is that he does not miss his wife, he misses his friend. Both key facts are unstated, they are assumed. Listeners will connect the dots that the writer omits. However, writers often need to write ideas in long form, before deciding the what to omit.

Takeaways

The expert advice on humor writing confirms that humor can be at least partially if not fully externalized. Experts expressed both high-level techniques such as explicitly violating

an expectation as well as low-level techniques such as using details or point of view. This indicates that any generative model of humor will require both high-level structure and low-level techniques.

5.3 Humor Analysis

The advice from experts gives us a good vocabulary and guiding principles for developing a humor writing process. However, it lacks details. To fill in the details, we analyze professional humor to see what mechanisms they repeatedly apply in their jokes.

Humor takes on many forms. This analysis focuses only one type of humor that is well-suited to analysis and experimentation: a section of *The Onion* called *American Voices*. *The Onion* is a popular source of humor among liberal college students. *The Onion* is most famous for its fake news headlines satirizing American culture. However, they also have a section called *American Voices* which starts with real news headlines and satirizes them by writing fake "average American" responses. For example, Table **??** shows one of the real news headlines and three fake responses to it written by *The Onion*.

American Voices is well-suited to our analysis because the headlines can be seen as an input to the system and the jokes can be seen as the output. The job of the analysis is to find the operations that connect the headlines to the responses. Because *The Onion* has three jokes per headline we know there are several paths of operations between a headline and a joke.

5.3.1 Finding Patterns in American Voices Patterns

We performed an analysis of 330 *American Voices* joke, in response to 110 real headlines. We found four general patterns:

Pattern 1: Multiple References

Jokes make multiple references to aspects in the headline. The references can be either direct or indirect. In the following joke, the joke makes four references to the headline.

Table 5.1: Example of the *American Voices* style of humor: a real news headline, a byline, and 3 jokes — fake "average American" responses.

Real Headline	Justin Bieber Baptized In NYC Bathtub
Real Byline	Pop star Justin Bieber was baptized in a friend's bathtub this weekend after weeks of Bible study and church services, with celebrity blogs reporting that the 20-year-old sought spiritual guidance in an attempt to wash away his sins following a scandal in which videos emerged of him using racial slurs.
Joke 1	"Oh my God! Can I lick the tub?"
Joke 2	"Great, now my teenage daughter's going to be begging me for $300 so she can reaffirm her devotion to God, too."
Joke 3	"Never let it be said that Bieber's PR people aren't bringing new ideas to the table."

Table 5.2: Example of the Multiple Connections pattern showing four concrete or oblique connections between aspects of the headline and things mentioned in the joke.

People Bending iPhones At Apple Stores	"I can't believe people would just walk into an Apple store and start breaking things like it's a Best Buy."
People	"I can't believe **people** would..."
Apple Stores	"...walk into an **Apple store**..."
bending iPhones	"...and start **breaking things**..."
Apple Store	"...like it's a **Best Buy**."

People Bending iPhones At Apple Stores

"I can't believe people would just walk into an Apple store and start breaking things like it's a Best Buy."

Two of the references are direct — "People" and "Apple Store" appear in the headline an the joke. In contrast, the other two references are indirect: The headline mentions "bending iPhones," and the joke references this as "breaking things." The headline mentions "Apple Store" and the mentions "Best Buy." Apple Stores and Best Buys are not the same, but they are clearly related. They are both electronics retailers — Apple Stores are at the high end and Best Buy is at the low end. Table **??** shows the four references in detail. Virtually all the jokes reference at least two aspects of the headline.

Pattern 2: Association Types

Indirect references in jokes fall into six prominent classes. We call these classes Association Types because they rely on the associative thinking skills of the joke writer. The "Apple Store" / "Best Buy" a type of association we call Alternative Thing because the two aspects are in the same domain, but are competitors or alternatives in that domain. Other examples

of Alternative Thing include:

> **UConn** Holding 'Football 101' Clinic For Female Fans
>
> *"Sure, I guess UConn's course is fine if you couldn't get into* ***Yale's*** *football clinic."*

> **Baskin-Robbins** To Honor Veterans With 'First Class Camouflage' Ice Cream
>
> *"This sure makes* ***Ben and Jerry's*** *look like a bunch of flag-burning ISIS sympathizers."*

Alternative Thing is one of six Association Types. Other Association Types are: Alternative Reaction (chicken prom corsage is expected to be gross/ a person would would want one), Details (SkyMall / useless home products such as 'Summer Savannah' Backyard Garden Lion Pedestal), Insult (Breaking Bad / having a terrible finale), Personality Flaw (George Clooney / obsessed, delusional stalker fan). Detailed examples including the headline and joke are in Table **??**.

Pattern 3: Belief Types

Associations explain many elements in jokes but an association is only part of a joke. In addition to associations, jokes contain a belief that the speaker holds about an aspect or association. For example, the belief in the Ben and Jerry's joke is that 'First Class Camouflage' Ice Cream" is bad for Ben and Jerry's because it makes them look unpatriotic. This belief type is called Reason Bad. Another Joke with a Reason Bad belief is as follows:

> McDonald's Testing Customizable Burgers To Compete With Chipotle
>
> *"Whatever happened to walking into a fast food restaurant, shouting a number, and eating whatever you were given?"*

Table 5.3: Examples of the Association Types pattern from our analysis of *American Voices* humor.

Association Type	**Headline**	**Headline Aspect**	**Association**	**Joke**
Alternative Thing	People Bending iPhones At **Apple Stores**	Apple Store	Best Buy	"I can't believe people would just walk into an Apple store and start breaking things like it's a **Best Buy**."
Alternative Person (Point of View)	**Justin Bieber** Baptized In NYC Bathtub	Bieber	Bieber fan	"Oh my God! Can **I** lick the tub?"
Alternative Reaction	KFC Selling **Chicken Prom Corsages**	chicken prom corsage	person who would want this	"My daughter's dress would look better with something from Burger King."
Specific Example	Report: **'SkyMall' Magazine** May End Print Edition	SkyMall	'Summer Savannah' Backyard Garden Lion Pedestals	"Alright, how many **'Summer Savannah' Backyard Garden Lion Pedestals** do I have to order to turn this thing around?"
Insult	Toys 'R' Us Pulls **'Breaking Bad'** Action Figures From Shelves	Breaking Bad	Finale was bad	"That's okay. My little guy **hated the finale**."
Personality Flaw	**George Clooney** Engaged	George Clooney	obsessed, delusional stalker fan	"**He usually tells me everything**, so I doubt this is true."

The belief in this joke is that testing new burgers is bad. The reason it is bad is because people prefer things the traditional way, not the new way.

Reason Bad is one of eight belief types. Other belief types include Reason Good, Effect and Solution. Examples are in Table **??**. "Effect" is the belief that an aspect of entity in the headline will have effect on the world or on the speaker of the joke. For example: .

> Scientists Developing Heat-Resistant Chickens To Withstand Climate Change
>
> *"How much to upgrade the chicken I already got?"*

The aspect 'developing chicken technology' could have an effect of requiring people to upgrade their technology.

The Solution belief finds a problem in the headline and proposes a solution to it. For example: .

> Report: 'SkyMall' Magazine May End Print Edition
>
> *"Alright, how many 'Summer Savannah' Backyard Garden Lion Pedestals do I have to order to turn this thing around?"*

The aspect of "Ending Print Edition" is a problem for this speaker and his solution is to buy more SkyMall stuff in order to "turn this thing around."

Pattern 4: Expectation Violation Types

From the humor theory and practice we know that expectation violation play a role in humor generation and appreciation. Violating Expectations is is an abstract goal, but our analysis of the *American Voices* found three concrete mechanisms for violating expectations.

Violation Type 1: Bait-and-Switch

In the Bait-and-Switch Violation Mechanism, the joke first grounds the listener in a statement that they expect to hear such as "I can't believe people would just walk into an Apple Store

and break things", but then abruptly adds a short phrase that changes the meaning entirely such as "like it's a Best Buy."

Here is another example of Bait-and-Switch:

> Facebook Lets Users Choose Who Controls Their Account After Death
>
> *"I'm not sure who I can trust with the solemn task of rejecting Bubble Witch Saga Requests."*

The bait is 'I'm not sure who I can trust with this solemn task',' and the switch is "rejecting Bubble Witch Saga Requests."

The Bait-and-Switch mechanism is consistent with Semantic Script Theory of Humor in that the bait is the first script and the switch is the second script.

Violation Type 2: Unexpected Angle

In the Unexpected Angle Violation Mechanism, the joke expresses an unexpected reaction to the headline by finding an unusual angle to interpret the headline through. Consider the following joke,

> Great White Shark Populations Surging Off East Coast
>
> *"It's an exciting time to be a shark, that's for sure."*

The expected reaction to the headline is that the surging shark population is bad because it creates a danger. However, that interpretation is from the point of view of a human who is afraid of sharks. From the point of view of a shark this is a good thing. The shark's point of view is the unexpected angle, and the unexpected belief is that this is a good thing. The belief that surging shark populations is a good thing violates expectations, which is why this joke is funny.

In the Camouflage Ice Cream joke, the usual angle is Ben and Jerry's. Ben and Jerry's is not directly mentioned in the headline, it is an Alternative Thing Association for Baskin-Robbins. The listener expects that releasing a new flavor of ice cream to honor veterans

is good, but it is bad when you consider the effect on the competitor: Ben and Jerry's. Considering the negative affect on an alternative aspect violates expectations.

Violation Type 3: Sarcasm

In the Sarcasm Violation Mechanism, the joke a false assumption in the headline by agreeing with it to a degree that makes it clear to the reader that the speaker is not serious. For example,

> Mick Jagger Blamed For Brazil's Historic World Cup Defeat
>
> *"There is absolutely no other explanation.'*

The headline contains the blatantly false assumption that Mick Jagger is to blame for Brazil's defeat. The joke agrees with this belief in such an exaggerated manner that it makes it clear the speaker is not serious, and thus brings the false assumption in the headline to the listeners attention. Sarcasm violates expectations in two ways: first, listeners do not expect people to say the opposite of what they mean and second, listeners are not fully aware of the false assumptions in the headlines thus highlighting them with a sarcastic response reveals an unexpected falsehood in the headline.

Here is another example of Sarcasm:

> Baskin-Robbins To Honor Veterans With 'First Class Camouflage' Ice Cream
>
> *"I look forward to placing a few of these on the graves of the fallen."*

This joke asserts that ice cream is a good way to honor veterans. However, this belief is sarcastic. Certainly in comparison with the traditional way of honoring veterans (placing flowers on graves), ice cream is an inappropriate way to honor veterans. The joke exposes the false assumption in the headline that Camouflage Ice Cream honors veterans.

Table 5.4: Examples of the Belief Types pattern from our analysis of *American Voices* humor.

Belief Type	**Headline**	**Headline Aspect**	**Belief**	**Joke**
Reason Good	Baskin-Robbins To **Honor Veterans** With 'First Class Camouflage' Ice Cream	Honor Veterans	This is good because ice cream is a great way to honor veterans	"**I look forward to** placing a few of these on the graves of the fallen."
Reason Bad	McDonald's **Testing Customizable Burgers** To Compete With Chipotle	Testing an improvement to fast food	This is bad because some people prefer things the way they are	"**Whatever happened to** walking into a fast food restaurant, shouting a number, and eating whatever you were given?"
Effect	Scientists **Developing Heat-Resistant Chickens** To Withstand Climate Change	developing chicken technology	The effect on people who own chickens is upgrading their 'technology'	"How much to **upgrade** the chicken I already got?"
Solution	Report: 'SkyMall' Magazine May **End Print Edition**	End Print Edition	A solution to going out of business is to buy Skymall stuff.	"Alright, how many 'Summer Savannah' Backyard Garden Lion Pedestals do I have to order to **turn this thing around**?"

5.4 System

5.4.1 Design

To simplify the problem of humor generation, we restrict ourselves to generating *American Voices* style of humor. The input to the system is a headline, and the output is a joke. As a further simplification, we use the same headlines that *The Onion* uses. There is a separate problem of selecting the headlines, that is left as future work.

We decompose the process of humor generation as a set of microtasks and a generative model of humor that uses those microtasks. The microtasks can be derived from the theory-inspired patterns found in the Humor Analysis section: breaking headlines into aspects, finding entity associations with aspects of the headline, defining new unexpected beliefs to express from the standpoint of associated entities. There are many possible ways to design a workflow that uses those microtasks. We first present two initial workflow designs that we

rejected before presenting the HumorTools workflow design.

5.4.2 Initial Design: Static Workflow

Our first design was a static sequence of microtasks people could apply to a headline to generate a joke. We defined the static sequence by selecting a joke and then back-engineering a sequence of microtasks needed to generate it. Here is an example of a joke and a back-engineered workflow for it:

> Report: Uber Adding 'Panic Button' For Chicago Passengers
>
> *"It's nice knowing Uber is willing to do everything it can to protect its customers short of properly screening its drivers.'*

Static Workflow:

1. Write down a expected reason this is good ("Uber is adding safety features for its passengers")
2. Pick an aspect of the headline ("Uber")
3. Write an insult about the entity ("Uber doesn't properly screen its drivers")
4. Combine the expected reason this is good from (1) with the insult from (3) into a Bait-and-Switch joke

When prototyping this workflow, we encountered two major problems. First, users were not always able to come up with the associations we wanted them to for new headlines or for the headline the pattern was derived from. An explanation for this is that users have limited world knowledge and it is unreasonable to expect them to come up with a specific kind of association for a given aspect. The system must allow enough flexibility for users to work with the limited world knowledge that they have.

Second, this static workflow did not generalize to many of the jokes we analyzed. Neither did any other static workflow we found. In fact, we found fewer than 10 jokes in the 330

that we analyzed that could be generated from the same static workflow. In order to write jokes that have the wealth of diversity that *The Onion* does, we have to be able to more flexibly apply the individual microtasks. The workflow cannot be static, it must adapt to the headline and to the users' knowledge.

5.4.3 Initial Design: Brainstorming

Our second design was to use microtasks to brainstorm associations with the headline that would lead to a joke. Given a headline and a set of 6 Association microtasks, users were able to apply at least one association to any headline. However, after brainstorming 10-20 ideas users were unclear on how to write jokes. Although they had many ideas, none of them constituted a joke. Additionally, when we pushed them to write anything joke-like, they had trouble. They reported being overwhelmed by the number of ideas and had no guidance on how to create a joke out of them. This workflow had the problem that many brainstorming sessions have — that the ideas are good, but it lacks a path to a concrete outcome. Although brainstorming is useful technique, a generative model with more scaffolding is needed to complete the joke writing process.

5.4.4 HumorTools Design

The descriptive theories of humor stress that expectation violation is the essence of a joke and this idea is born out in the three concrete mechanisms that *American Voices* uses in its jokes. One reason jokes are hard to write is because it is not trivial to violate someone's expectations using these mechanisms. Thus, in the HumorTools generative model, fulfilling one of the 3 expectation violation mechanisms is a constraint that must be satisfied. To help satisfy this constraint, the associations define a search space on concepts characteristic of *American Voices* humor. Multiple associations may be applied to satisfy the constraint, thus most microtasks can be chained together. Not all chains of microtasks are guaranteed to result in a joke. Thus HumorTools enables users to explore a tree of associations stemming from the headline, and they can backtrack when they get stuck.

5.4.5 Implementation

HumorTools is a web application implemented in Meteor. It defines 20 microtasks and teaches them each in isolation by showing examples of professional headlines and joke with pieces missing, and asking the user to fill in the gaps. Users are not expected to get the same answers as *The Onion*, but they can see what *The Onion* wrote and self-assess whether they are thinking along the same lines.

Although microtasks are independent, they are presented in an order that emphasizes the patterns in which they are typically chained together. Identifying headline aspects is almost always first. This is typically followed by either writing an expected reaction to the headline, or by writing an association to an aspect of the headline. Belief microtasks typically follow from an Alternative Thing or Alternative Person Association. Unexpected Angle violation mechanisms can often be constructed from the belief of an Alternative Person Association.

When writing jokes for headlines, users apply microtasks at their own discretion. HumorTools support the writing process by structuring their exploration of the conceptual space in a tree. This facilitates backtracking, keeping multiple threads open, and knowing which aspects of the headline have not yet been explored. HumorTools instructs workers to record the constraints they met in their final joke. This gives them a heuristic for when they are done, and lets them know what they are missing and must keep working to complete.

5.4.6 Microtasks

HumorTools teaching users 6 types of Microtasks, some with many subtypes of Microtasks. An example of the interfaces uses to teach the microtasks are given in Figure **??**. In this section we detail the input and output of each microtask and provide examples.

Identify Aspects

In the Identify Aspects microtask, the user in given a headline, and the goal is to list individual aspects of it. For example, in the headline:

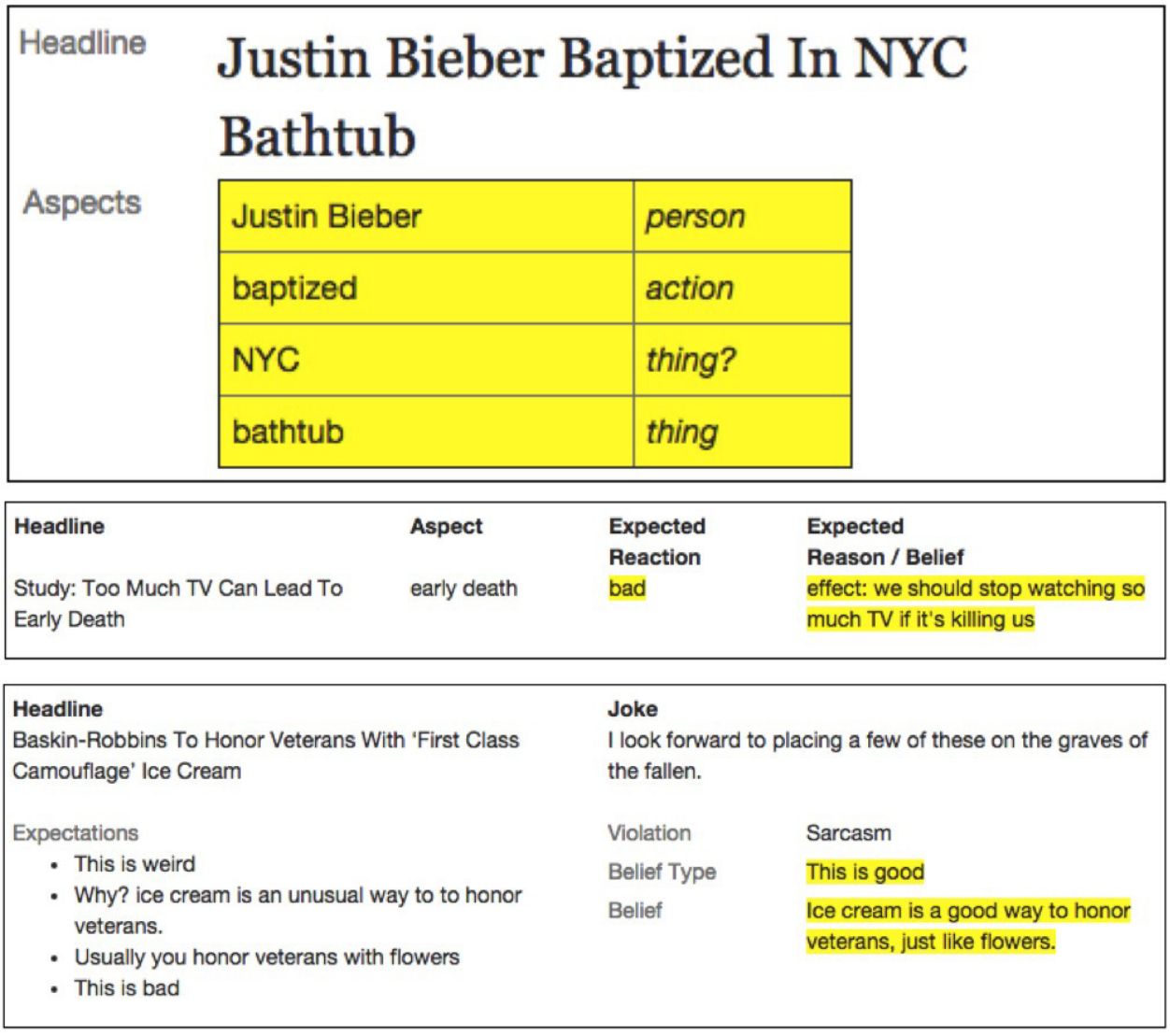


Figure 5.1: Three exercises from the microtask tutorial. The highlighted text is typed by the user. Microtasks from op to bottom: Aspect, Expected Reaction and Reason, Violation.

"Justin Bieber Baptized in NYC Bathtub"

Aspects include: "Justin Bieber", "Baptized", "NYC", and "Bathtub."

This simple task is an easy way to start writing a joke for a headline, and helps expand the headline into pieces that can be reused in the joke. Some aspects will be used directly, and some will undergo transformations from other microtasks.

Expected Reactions and Reasons

In the Expected Reaction and Reason microtask, the user chooses an aspect, and writes down whether they expect the reaction to be positive or negative and why. For example, one reaction to "Justin Bieber" is that he is bad. Possible reasons include disliking his music or that he is a bad role model.

Jokes are characterized by violating expectations. Thus, writing down expected reactions makes it clear what the expectations the user is going to violate are. Giving a reason helps expand an intuitive reaction into more material to find opportunities to violate the expectation.

Aspect Associations

In an Association microtask, the user chooses an aspect and writes down an association with it. For example, "Justin Bieber" is associated with "obsessive fans." People's associative memory makes brainstorming associations fairly easy. However, there are many possible ways to associate things, and some of them may be more useful to humor generation than others. In the analysis of *The Onion*, we identified six types of associations that *The Onion* uses repeatedly in their jokes. The "Justin Bieber" / "Justin Bieber fan" association is an Alternative Person type of association. Other Alternative Person associations are "Justin Bieber public relations people" or "Justin Bieber fan's parent."

Associations help expand the context of the headline to find more opportunities to violate expectations. Alternative Things, Alternative Person, Alternative Reaction are especially

helpful for finding point of view, or an angle to fulfill an Unexpected Angle expectation violation.

Beliefs

In a Belief microtask, the user takes an aspect or associated aspect, and writes down a belief about it that is different than the expected reaction or reason. Here are examples of how three different Alternative Person associations can lead to three different Beliefs: A Justin Bieber fan would believe his baptism is a good thing, which is counter to the expectation that this it is bad. A Bieber fan's parent would think it is a bad thing, but not for the expected reason — but because of the negative effect on them ("Great, now my teenage daughter's going to be begging me for $300' so she can reaffirm her devotion to God, too.'). Lastly, from the standpoint of Bieber's public relations team, this is good because it shows they are clever.

Beliefs are needed to take an association and make a story around it that will violate an expectation. This is consistent with the Semantic Script Theory of Humor that claims jokes need to have an expected script and an opposing script. Beliefs help users generate the opposing script.

Beliefs are fairly abstract, and like associations, there are many possible beliefs. From our analysis of *The Onion* we found eight Belief Types that were repeatedly used in jokes. The two most intuitive Belief Types are Reason Good and Reason Bad. Effect on the Speaker and Solution are two less intuitive but useful. See section **??** and Table **??** for examples. Other Belief Types include Effect on Others, Blame, Predict Fail, Sympathize.

Expectation Violation

In an Expectation Violation microtask, the user connects pieces of information that complete one of the three Expectation Violation Mechanisms in section **??** and identifies the type mechanism which they are using and fills in specific details about that type of mechanism.

Because of the complexity of this microtask, we only taught users two of the three violation mechanisms we found: Unexpected Angle and Sarcasm.

In Unexpected Angle, users identified the angle (e.g., SkyMall shopper), Belief Type (e.g., Solution) and the Belief (e. g. That buying more SkyMall stuff will solve the problem).

In Sarcasm, users had to identify the hidden false assumption, the Belief Type, and the Belief. Consider the following joke:

> Poll: Elite Colleges Don't Produce Happier Graduates
>
> *"Is there no joy in this world for the privileged?"*

The hidden false assumption is that news is bad. The Belief Type is Sympathize. The Joke goes along with the false assumption that this is bad news by sympathizing with the elite. The Belief that expresses that sympathy is the exaggeration that there is no joy for the elite.

Heuristic Evaluation

It is difficult to know when a joke is is complete. We declare a joke complete when it passes the heuristic evaluation containing three things:

1. An Expectation Violation Mechanism (e.g. the Unexpected Angle Expectation Violation Mechanism)
2. A belief (e.g. the belief that Bieber's baptism is a good thing)
3. Two references to the headline (e.g. Bieber/Bieber fan is the first reference, and Bathtub/tub is the second reference)

Once the user writes their joke, they fill out a small form detailing a heuristic evaluation of the joke. For example, in the Bieber bathtub joke, the Expectation Violation mechanism is Unexpected Angle, the Belief is that Bieber's baptism is a good thing, and the two references are "Bieber" / Bieber fan (represented as the speaker of the joke, and "Bathtub" / "tub".

If any of these items are not in the joke, the user must backtrack and continue working. This heuristic evaluation is what ensures that the constraint has been met. Previously, we described that the jokes were about satisfying a constraint. That constraint actually has two parts: one is violating an expectation, and the other is containing two headline references. Satisfying the Expectation Violation constraint is much harder, so for simplicity we say that is the major constraint. Usually the second constraint is met by simply applying both an Association and a Belief. The Expectation Violation Constraint is inspired by the humor literature, the Two Reference Constraint comes from the analysis of *The Onion*, but some comedians mentioned that connections are a part of what makes something funny.

5.4.7 Adaptive Workflow

Once users have been trained in the microtasks, they can start with a headline, explore a conceptual search and find an Expectation Violation. Users have must be allowed freedom in their choice of microtasks in order to adapt to the current context and to the limited to their knowledge. However, the search space is large and there are three techniques we use in the adaptive workflow to help guide the search and manage the exploration.

Although the microtasks are independent, there are some implicit constraints on their order. The first microtask is always to identify the aspects of the headline. This gives users a concrete way to start the joke writing process and sketches out the different branches to explore. Once aspects have been identified, it is typical to apply Association and Expectation microtasks to them. Additionally, there are pairs of microtasks where one microtask naturally transitions into the other. For example, Alternative Person Associations often leads to an Unexpected Angle Expectation Violation Mechanism. These transitions are not explicitly taught to users, but implicitly implied in the order of the microtasks we teach and how examples build on one another.

As a measure against users being overwhelmed by the size of the search space, HumorTools instruct users to follow their train of thought and if they get stuck, backtrack. People are much better at following one train of thought than jump between different contexts. However,

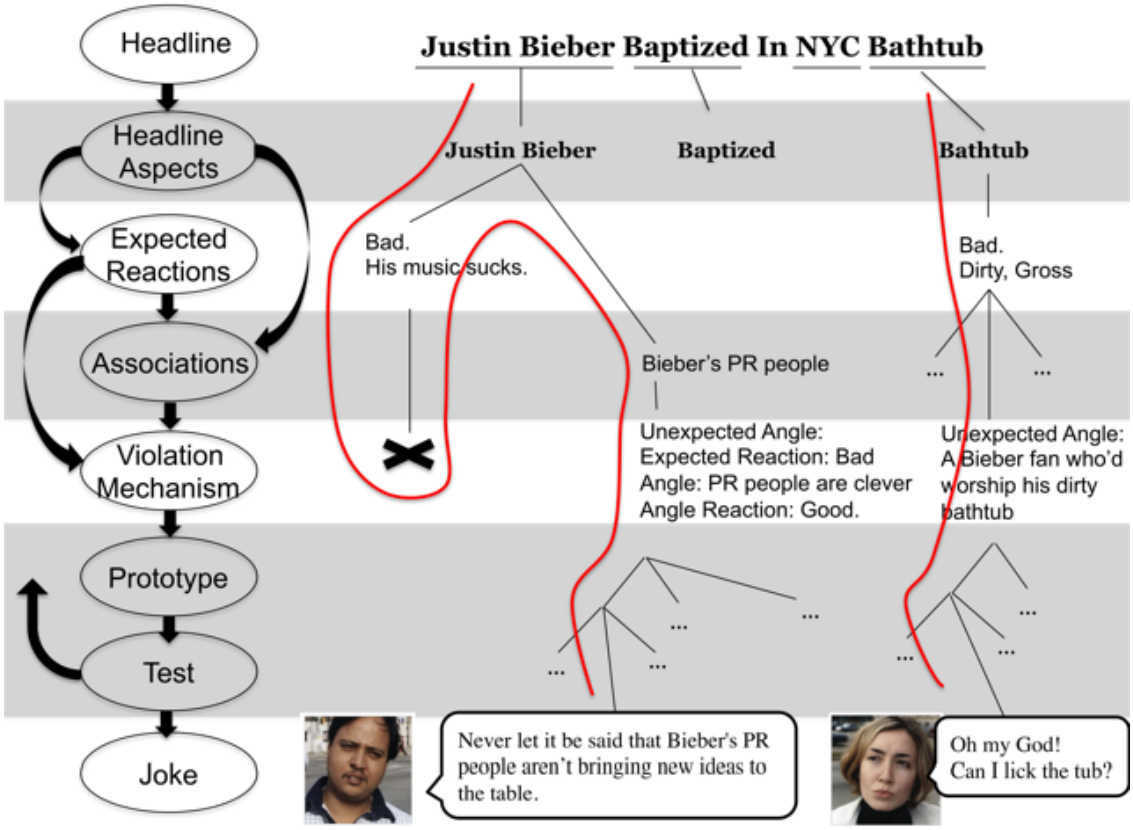


Figure 5.2: An illustration of the HumorTools adaptive workflow generating two jokes — one with backtracking, one without. The left side shows the typical transitions between types of microtasks, the right side shows the search paths through the conceptual search space from the headline input to the joke output.

people also have tendency to follow one train of thought too far. Although this is still a danger in any system with people in the loop, HumorTools' design of writing out a branch explicitly makes it more attractive to backtrack and take a difference approach when the user gets stuck.

Users of HumorTools use the Heuristic Evaluation microtask to know when their joke is complete. This provides a stopping condition for the workflow. The Heuristic Evaluation looks for three things: an Expectation Violation Mechanism, a Belief, and Two References. It is up to the user to identify these aspects of their joke. If they can, they they are done, if not, they have to backtrack and either add to their joke or start a new one.

An illustration of the HumorTools search process is depicted in Figure **??**. On the left are the key types of microtasks, arrows between the microtasks represent typical transitions. The right side shows the pathways for two jokes — one with backtracking and one without. Red lines indicate the two paths. The process starts by identifying aspects in jokes. For the first joke, starting with the "Justin Bieber" aspect we write the Expected Reaction (that he is bad) and a reason ("his music sucks.") In this thread we did not see an opportunity to apply a violation mechanism, so we backtrack. Going back to Justin Bieber, we see opportunity to apply an Alternative Person Association. An overlooked person in this scenario are Bieber's public relations team. From this Alternative Person Association there is a possibility to apply an Unexpected Angle Association. We expect this headline to be bad, but from the point of view of Bieber's PR team, this is a good thing because they are doing their job. A belief we can apply to to Praise Bieber's PR people for being clever. The joke here is "Never let it be said that Bieber's PR people aren't bringing new ideas to the table."

The HumorTools Adaptive Workflow is not a deterministic process. Multiple jokes are possible. Another joke stems from the Expected Reaction to the Bathtub being dirty and gross, but the Unexpected Angle is that a fan would not care and would worship him anyway. The joke is "Oh my God! Can I lick the tub?" (Note: these jokes were written by *The Onion* to illustrate in an abbreviated fashion how these jokes could have been made. The Evaluation section has examples of jokes made by users.)

5.5 Evaluation

5.5.1 Setup

To evaluate HumorTools, tested if humor novices could use the microtasks and workflow to create *American Voices* jokes. From our early investigations, we knew that people can write *American Voices* jokes, but do so by intuition and with a limited range of techniques. Often they get stuck after one joke. Our aim was to teach people to write jokes using an externalized process. If that is possible, we want to know which elements of the workflow are most helpful to users, and how funny those jokes are compared to *The Onion*.

5.5.2 Study

We designed a web-based study lasting between 60-80 minutes that participants completed at home. We advertised the study through mailing lists on a large college campus. Participants were paid $20 for their participation. The stages of the study were as follows:

1. Rate *American Voices* jokes. This gives users a warm-up and 24 examples of the target style of humor. This lasts about 5 minutes.

2. Write jokes intrinsically. We give participants 5 minutes to write jokes for 3 headlines. This gives users a chance to experience joke writing without the tutorial so they will later be able to compare their experiences. (referred to as the Intrinsic stage)

3. Do the tutorial. The tutorial on the microtasks and workflow. It is self-paced and lasts 30-50 minutes.

4. Write jokes with the HumorTools Workflow. We give participants 15 minutes to write jokes for the same three headlines as in stage 2. This gives users a chance to write compare their intrinsic joke writing process to the HumorTools workflow.

5. Survey. An exit survey asking about their experience and demographics. This lasts about 5 minutes.

In our study design, the Intrinsic joke writing step was only 5 minutes whereas the HumorTools writing steps is 15 minutes. These parameters were established in earlier prototypes. When writing joke intuitively, people do not need much time. However, in the HumorTools workflow, writing steps out takes time, so we allot more time in HumorTools to create a fair comparison.

In the Intrinsic and HumorTools joke writing stages, we used the same headlines as prompts. This means that when the see the headlines in the HumorTools writing session, they are already familiar with the headlines and have already written a joke intuitively for them. In some sense they are biased by already this familiarity, however that is not a problem for this study. First, it is good that they have already written a joke intuitively for the headline. In an non-experimental setting for joke writing using HumorTools, we do not expect to preclude people from writing jokes intrinsically. We expect that users would write jokes the intrinsic way until they ran out of ideas, then switched over to HumorTools. Thus, having already written intuitive jokes is consistent with intended practice. Second, using the same headlines in the Intrinsic and the HumorTools writing sessions is the only way to compare the two methodology. Because of the dynamic nature of joke writing, the choice of the headline determines much of the experience. People find some headlines harder or easier for reasons yet unknown to us. We want to make sure the difference in experience is due to the tools and not the headline choice. Randomizing for headlines is part of our future work.

The jokes were rated by the authors as either below the bar or above the bar for joke style and quality consistent with *The Onion*. Nine of the good jokes were selected by the authors to be rated for funniness against the *The Onion* by an audience of readers who enjoy *The Onion*. In this study, raters see a headline and six jokes for it. Three are written by *The Onion*, and 3 are written by HumorTools users. For each joke, the raters rate each joke by answering whether it is funny: "yes," "no," or "meh."

5.5.3 Results

The humor writing study had 20 participants (8 female, 6 male, 6 declined to say) all between the ages of 18 and 25. The humor rating study had 21 participants, all of whom where college students who were fans of *The Onion*.

HumorTools Completion

The HumorTools process is novel and has many new concepts for users to learn in the tutorial. The biggest danger in HumorTools is users not completing the process. Across the twenty users in the study, we found that 75% of users were able to write complete jokes in HumorTools. 25% were above our quality bar. Of the 9 funniest jokes written in HumorTools, 3 were rated funnier than a professionally written Onion joke. See Figure **??**. Although we do not know how much time *The Onion* put into their jokes or how many the rejected before deciding to print these, this represents a promising future for novice externalized humor generation processes.

User Experience

We asked users to compare their intrinsic humor writing approach with HumorTools. We wanted to know which they preferred and what was most helpful to them.

Of the 20 participants, 17 (85%) said that HumorTools was helpful to their joke writing process. Three (15%) said that the workflow was not helpful and they preferred to write jokes intrinsically.

For the participants who liked HumorTools, 6 of the 17 described their the workflow make joke writing more structured, systematic, or methodical. 5 of the 17 described their process before as being *"instinctual"* or they ‘*"didn't think about it"* or in one case their process before was to ‘*"[let] things sit a lot more in my head and waiting for a joke to magically pop out"* (p5). Three participants said the workflow was helpful in getting started writing jokes.

The participants who reported disliking the workflow all had different reasons. One

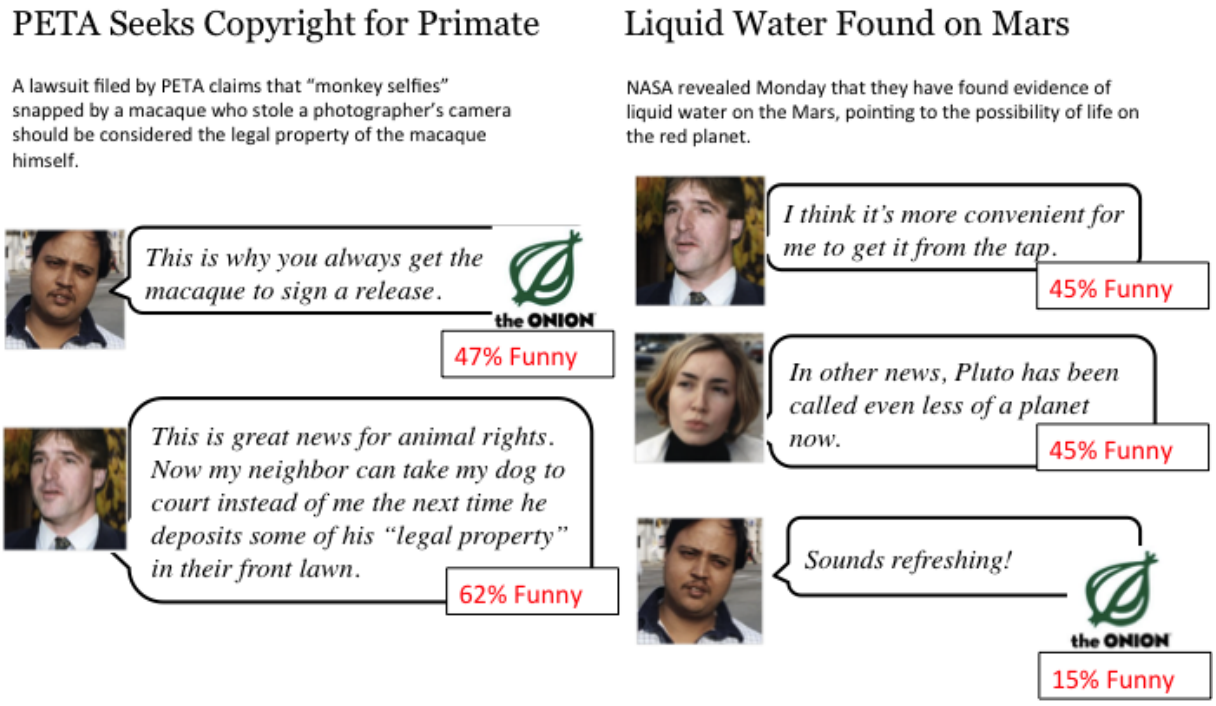


Figure 5.3: Two headlines with jokes from both HumorTools and *The Onion*. These are 3 instances where HumorTools jokes were rated funnier than *The Onion*. The two jokes written by *The Onion* have *Onion* logos on their far right.

thought it was better to just learn by example and and another said it was over-simplified and *"only for beginners."* In earlier pilots, we ran HumorTools on two users who had written professional or semi-professional humor. Both reported that HumorTools did nothing for them. One said it was easy enough to do intrinsically, the other said,"This is basically similar to the approach I use in my head, but [HumorTools] slows me down." These are valid reasons not to use a workflow.

The two microtasks participants reported most helpful were Associations (10 of 17) and Violations (10 of 17). Aspects were mentioned by 4 participants, Expected Reactions were mentioned by 4 participants, Beliefs by 2 participants (all out of 17 participants who found the microtasks useful). This wide spread is probably product of users' variety in innate humor writing abilities.

Associations were useful because they gave participants more possibilities for ideation. In their words, *"[Associations were] helpful for finding a new angle or a way to be sarcastic"* (p14) and *"[Associations] helped me think of jokes in a wider conceptual space than I previously had."* (p9) With and without the workflow this microtask was helpful: *"[Associations] generate a bunch of great ideas for a joke and makes it much easier to finish them with either the other techniques or just by thinking."* (p4)

Violations were useful because it make the actual mechanism for the joke more concrete and externalized. *"I used the Violations because I knew that I had two choices — helped to narrow it down and make it feel manageable"* (p14). Particularly, of the two Violation Types — Angle and Sarcasm — Angle was mentioned by 4 times participants who benefited from Violations. Their natural inclination was toward sarcasm and adding angle as a second mechanism enabled them to land far more jokes. Even for people who already had natural sarcastic tendencies, other microtasks enhances their sarcastic abilities when put together in this workflow. *"Expected Reactions and Expected Beliefs [were useful] because they helped me to be sarcastic more intentionally — I could use them to make sure the jokes violated expectations"* (p14).

All the microtasks were named as useful by at least one participant. It is not surprising

that participants found different things useful. We attribute this to their difference in inherent abilities to write humor. It is hard to know which microtasks will unlock possibilities for people and which ones they either do not need or already know. In the future, being able to rapidly assess this would make teaching faster and more effective.

Areas for Improvement

We asked participants for areas of improvement. Four people found the tutorial too long and this probably impaired their ability to write jokes at the end due to exhaustion. Six people found one or more of the microtasks in the tutorial obvious. It is future work to teach the HumorTools microtasks more iteratively and more interactively.

5.6 Discussion

HumorTools is a first step towards a system that makes humor generation externalized. HumorTools is decomposes humor into microtasks and externalizes the process, but does not yet crowdsource the generation of humor. In the current study, one person writes a joke from start to finish. We believe that by decomposing humor generation into microtasks, we open the possibility of collaboratively writing jokes where multiple users can search the conceptual space of jokes together and build on the associations of others.

There are several ways to continuously improve the quality of the HumorTools jokes. The current set of microtasks is sufficient for writing humor, but we may be missing critical microtasks that *The Onion* uses to be funny. In fact, this is definitely true because we only taught three Expectation Violation Mechanisms found in *The Onion*. We can continuously improve the HumorTools workflow by comparing HumorTools jokes against *The Onion*, finding cases where *The Onion* is better, and analyzing those jokes for microtasks not yet in the tool.

HumorTools currently focuses on one type of humor, and it is future work to generalize the techniques to other forms of humor. Another form of humor that has input/output pairs is *The New Yorker* Caption Contest. *The New Yorker* has a different style of humor than

The Onion. To extend HumorTools to this new domain, we would need to analyze examples from *The New Yorker* to find what types of associations, beliefs, and expectation violation mechanisms they use. We can also possibly extend to forms of humor that do not have input/output pairs. Most of *The Onion* is satirical headlines — for which we have no input. However, we hypothesize that every joke has some kind of input or inspiration. Often it is hidden or diffuse, but by looking at many examples of *Onion* headlines, we could possibly back-engineer the types of targets that inspire their humor.

We believe that posing creative tasks as a constraint-satisfaction problem within a conceptual search space can be generalized to other domains. One such domain is writing persuasive argumentation, such as op-eds in newspapers. Extending this approach to other domains requires defining the constraints and search spaces that characterize the domain. This can be found by look at examples and drawing from theory. For persuasive argumentation, existing op-eds can serve as the examples and the study of rhetoric can provide clues as to what the constraint and search spaces might be. The most important constraint to meet in argumentation is to have a thesis that can be supported by arguments. This is generally how argumentation is taught. To find out what is persuasive, we could look at metaphors, tone, and the use of story narrative to make data fit into a compelling story.

5.7 Summary

Many people assume that humor requires the magic of creativity, and that it cannot be broken down and made systematic. HumorTools presents an externalized process for generating humor that relies on human intelligence, but makes it easier to write humor. Humor generation, and creativity in general, probably cannot be reduced to a static, cookie-cutter approach. The generative model we propose is a constraint problem that can be satisfied by searching a conceptual space of aspects, associations and beliefs. The HumorTools model was informed by theories of humor which characterize humor as violation of expectations, and by analysis of examples to find concrete mechanisms for violating expectations as well as microtasks that define the conceptual search space.

In a 20-person study of HumorTools 75% of users were able to generate humor in this externalized way. In a survey and interviews, users reported liked that it helped them start writing jokes in a concrete way, and gave the enough guidance to make the process more concrete. It also helped them create a wider variety of jokes than their internalized process did. In a comparison against *The Onion*'s jokes for the same headlines, three of the HumorTools jokes were rated funnier than *The Onion*. We believe this approach to decomposing creative problems into an externalized approach can generalize to other domains, lead to better ways to teach creativity, enable better collaboration, and result in more systematic generation of creative artifacts.

Chapter 6
CONCLUSION AND FUTURE WORK

This dissertation has shown how to systematically solve open-ended problems with adaptive crowd algorithms. This final chapter recapitulates the contributions made by the presented systems, and concludes with an outlook on future work.

6.1 Restatement of Thesis and Contributions

Adaptive crowd algorithms use crowdsourced microtasks to solve problems by starting with the concrete context of the problem and adaptively iterating towards the goal. The microtasks give workers partial information about the problem, and the results of the microtasks are proposals for partial solutions. *Adaptive mechanisms* use partial solutions to iterate towards the goal by exploring multiple ideas, testing ideas, adapting to feedback and nudging the solution forward into a cohesive whole.

To demonstrate my thesis, I introduce three systems that systematically solve open-ended problems with adaptive crowd algorithms:

- Cascade crowdsources the open-ended problem of taxonomy creation.
- Frenzy coordinates a crowd of experts to meet the constraint of organizing accepted conference papers into thematic sessions.
- HumorTools decomposes the creative task for writing humorous news satire in the style of *The Onion*.

The algorithms embodied in these systems introduce several concepts and techniques. The first is the concept of adaptive crowd algorithms, which is a new approach to crowdsourcing that uses adaptive exploration rather than static workflows to combine the work of

individual contributors. Second, these systems uses three microtask primitives: Generate, SelectBest, Categorize. These primitives were used in Cascade and Frenzy and can be reused in other crowd algorithms. Lastly, these systems each introduce a novel adaptive mechanism. Cascade introduces the recursive generate-and-test mechanism that achieves the global-from-local property that each worker sees only a local view of the data, but the algorithm produces a globally cohesive output. Frenzy uses flare-and-focus to brainstorm many possible labels before trying to find a set that satisfy a difficult global constraint. HumorTools searches a conceptual space with backtracking until a constraint can be satisfied. My evaluation shows that with adaptive crowd algorithms, we can solve open-ended problems too big for one person, too ill-defined to automate and that require creativity.

6.2 Limitations

Although we believe that many creative problems can be solved with adaptive crowd algorithms, it is possible that some cannot. Adaptive crowd algorithms rely on the workers in the crowd to have shared terms and knowledge of design patterns. However, some exceptionally novel solutions break the mold by not using traditional design patterns. Some solutions invent new techniques. For example, James Joyce's Ulysses [**?**] abandoned traditional prose structure for a new stream of consciousness style. Similarly, to prove the Halting Problem is undecidable, Alan Turing invented a new mechanism – Turing Machines [**?**]. Problems that require inventing new mechanisms are outside the scope of adaptive crowd algorithms.

Although adaptive crowd algorithms can solve problems too big for one person, size may be a limiting factor. Even for complex problems with design patterns, such as a cliched romantic comedy screenplay, it may be easier for one person to do all the microtasks simply because of the amount of context needed in order to contribute. Large tasks accumulate context: characters names, backstories, motivations, and constraints on the solution learned from previous ideas that failed. Once a person has learned the context of the problem, it is could be cheaper and easier to reuse the same person rather than teaching that context to a new person. This makes crowdsourcing the problem impractical. Instead, we could

either try consciously dividing the problem to limit context needed, or pivoting away from crowdsourcing and towards selfsourcing [**?**]. This would help a single user complete a complex task in a systematic way rather than using the crowd.

6.3 Future Work

HumorTools is the first step towards the ultimate goal of systematically generating creative artifacts. In the future, the HumorTools approach can be extended to other forms of humor and to new domains of creativity. The immediate next steps are to build on the existing HumorTools platform by crowdsourcing the process, and increasing the quality of the jokes. In the near term, I can begin to automate some or all of the microtasks in the process. After that, HumorTools can be extended to other styles of humor such as *The New Yorker*'s caption contest and the satirical headlines that *The Onion* is traditionally known for.

Further in the future, we can generalize the HumorTools approach to other domains. We propose two domains: writing persuasive argumentation and conducting empirical economics research. Additionally, to enable people to decompose other creative domains, we need tools to support the decomposition process. This process is similar to sensemaking, and the tool we used to support the humor decomposition/sensemaking process was Frenzy [**?**]. We can build on the insights of Cascade [**?**] and Frenzy [**?**] for crowdsourcing data organization to design tools that support distributed sensemaking. This will enable us to crowdsource not only the generation of creative artifacts in chosen domains, but also the process for decomposing new domains of creative artifacts.

6.3.1 Extending HumorTools

Choosing headlines

HumorTools currently uses real news headlines as input and outputs fake man-on-the-street style jokes. However, the real news we used in the evaluation are not randomly selected news headlines, they are the same headlines that *The Onion* uses in their *American Voices*

section. These headlines have clearly gone through a selection process. Compared to the average headline, they are probably easier to make fun of. Some of them are already funny, most of them are not overly negative, and there are probably features of these headlines that make them prime for writing *American Voices* style jokes. In future work, HumorTools could also back-engineer the process of selecting the headlines.

Finding more constraints and search spaces

HumorTools frames the process of writing jokes as a constraint satisfaction process. Currently, there are three constraints: the joke must contain an expectation violation, the joke must have two connections from the headline to the joke, and the joke must contain a belief. These are the hard constraints that we know about, but there are probably also soft constraints that we have not yet discovered. Based on the results so far, additional constraints might be that the joke not be overly offensive, or that it should be as short as possible. These might be soft constraints because there are clearly examples of offensive jokes and jokes that are funny because they are too long. In future work, we can find additional constraints by identifying jokes that aren't funny and attempting to fix them. If a joke is negative and not funny, does making it less negative fix it? If a joke is too long, does making it shorter improve it? If so, then these additional constraints should be added to HumorTools.

In HumorTools there are are seven types of Association microtasks (Alternative Person, Point of View, Insult, etc.). These seven types do not represent the only types of association, there are probably many more. Further analysis could identify others. Additionally, involving more people with different perspectives could help identify even more. We also do not know the frequency of each association type or the frequency with which two microtasks co-occur. A frequency analysis could show that, for example, that the Alternative Person microtask is frequently paired with the Effect on Others belief. For example,

> Headline: *Study: Firstborn Children Have More Ambition*
>
> Joke: *"I knew there had to be a scientific reason my sister has a job and I don't."*

The speaker is a secondborn child, which is an alternative to "Firstborn," and this job considers the effect on the speaker. If such co-occurrence patterns exist, HumorTools could suggest which microtasks to apply to optimize the search process for users. By finding more microtasks and optimizing the search process, we can increase the quality of jokes.

Crowdsourcing humor

Currently, HumorTools uses one person to complete the microtasks from the headline to the joke. It would be further indication that HumorTools decomposes the process of writing humor if multiple workers completed different microtasks for the same joke. One reason to believe that this is possible and perhaps even more successful than the current approach is that some comedians work in writers' rooms. In a writers' room, multiple comedy writers sit around a table and make suggestions. Writers build off the suggestion of others to make jokes they might not have otherwise come up with. There is literature on brainstorming that indicates that this is possible for associations, which is one of the steps of writing humor. Future work can determine what microtasks are most suitable for crowdsourcing.

Improving humor quality

As users make jokes with HumorTools and we rate the jokes against *The Onion*, we will have more data on how funny HumorTools is and what patterns are the funniest. Moreover, by analyzing the jokes where *The Onion* is still funnier than HumorTools, we can look for microtasks that we might have missed. By continuing to compare ourselves to *The Onion*, analyzing our mistakes and improving the process, it is possible that HumorTools will be consistently funnier than them.

Automating humor

One of the benefits of microtask crowdsourcing is that the results of the microtasks can create training data for machine learning algorithms. With enough training data, it might

be possible to automate the microtasks. Many of the microtasks in HumorTools could probably be done by an algorithm already. Identifying aspects of a headline could be done by automated entity extraction, judging sentiment could be done by sentiment analysis, and WordNet may be used for some types of association. Future work can investigate automating some or all steps of the HumorTools process and find ways to use the human inputs to improve automation where it is needs help.

Extending to new forms of humor

HumorTools currently focuses on only one specific type of humor. The microtasks and adaptive workflow for generating humor are designed specifically for *The Onion's American Voices*. A first step towards generalizing the HumorTools approach is to apply it directly to a different style of humor and evaluate how well it works. The caption contest for *The New Yorker* cartoons is a good first target. *The New Yorker* style of humor is different from *The Onion* but the caption context and American Voices have one important structural element in common: they both have a concrete input. Because of this structural similarity, HumorTools can be directly applied. *The New Yorker* caption contest's input is a cartoon without a caption, and the goal is to generate a funny caption for it. Users could apply the HumorTools workflow by first identifying aspects of the image, coming up with associations and expectations, then trying to find an expectation violation. For captions, there may be an additional constraint that the caption must be from the point of view of someone in the cartoon.

We expect that because of stylistic differences between *The Onion* and *The New Yorker*, some of the microtasks may have to be changed. By analyzing examples of *The New Yorker* captions, we can identify new microtasks that *The Onion* does not use. Many of these will hopefully fit in the existing framework as new association types or new belief types. But it is possible we will find an entirely new type of microtasks that we did not identify in our analysis of *The Onion*.

We expect that some microtasks such as the insult association which we found in *The*

Onion will also be used by *The New Yorker*. However, the examples of the insult association microtask training may have to change to reflect the style of *The New Yorker*. *The Onion* may have more examples of insult jokes that target idiotic Americans whereas *The New Yorker* may have examples of insults that target neurotic, intellectual Woody Allen-types. To extend HumorTools to *The New Yorker*, we will have to find new microtasks and new examples, but hopefully the general workflow of meeting a constraint by finding violating an expectation will be the same.

A further challenge is to extend HumorTools to jokes for which we do not know what the input was. *American Voices* takes a real news headline as input, but for the rest of *The Onion*. we do not know what the input was, or if there even was one. Our prediction is that every joke is the result of some input. Comedians refer to it as their inspiration. In the case of *American Voices*, the inspiration is concrete. For other humor in *The Onion*, the inspiration may be less obvious. It may not be a single headline but a collection of observations the writer has made throughout their lifetime such as the annoying waywardness of shopping cart wheels. One way to approach writing jokes for headlines with unknown inputs is to analyze humor from *The Onion* and try to back-engineer what the input or inspiration was. There could be classes of inspirations such as observation of things that are annoying or noticing hypocrisy in politics. If we find classes of inspiration, then we can add an additional step to HumorTools to search for observations that can be turned into inspirations.

6.3.2 Decomposing Creativity

After exploring humor more deeply, the next step for decomposing creativity is to apply the HumorTools approach to other creative domains. The general HumorTools approach is to define a constraint that encodes what is difficult about generating the creative artifact and establish a search space for producing the building blocks necessary to meet that constraint. Defining the constraint is hard. Instead of starting from scratch, the HumorTools approach was to read the literature from philosophers, linguistics, and comedians and look for common themes. The literature contained many insights on how to describe the process in the

abstract. But to concretely establish a search space for the building blocks, we looked at examples. This approach worked for humor, and to demonstrate its generality there are two creative domains that we can next target for decomposition: persuasive argumentation and empirical economics research.

Persuasive Argumentation

Persuasive argumentation is like humor in that it is creative and subjective yet still has patterns and structure. The field of rhetoric has been analyzing and describing the building blocks of argumentation and reason since the days of Ancient Greece. The difficult constraint to meet in persuasive argumentation is to have a thesis statement that can be supported with arguments and that can be defended against counter-arguments. There are many rhetorical devices — such as metaphor — can that be used to express an argument. Some examples exist on websites that explain rhetoric, but we can extend these by mining opinion editorials from newspapers for concrete instances of the building blocks of arguments. By decomposing persuasive argumentation, we can crowdsource the creation of opinion editorials, have them judged by experts and submit them for publication to newspapers.

Empirical Economics Research

The HumorTools approach, like design, is iterative and involves making observations about a situation, testing ideas, and backtracking when the the current thread of ideas does not meet a constraint. In the future, it may be possible to apply this systematic approach to scientific research. To test this idea, we should start in an experimental field that does not need physical equipment. Empirical economics research is an ideal field. The constraint that economics research needs to satisfy is to find a research question, a data set, and an evaluation metric such that the evaluation metric can be used to test the chosen question on the data set. For example, using data from players scores from professional golf tournament pairings as a naturally randomized experiment to test whether competition has a positive effect on individual performance [**?**]. The current approach to research in economics is to let

individuals work alone or in small groups to creatively search for a set that works together. However, if instead we made a publicly available list of research questions, data sets, and evaluation metrics that was searchable and could be annotated, we could search for a set that satisfied the constraint more systematically. This search could be conducted by many researchers in parallel, building on the work of one another. Such a systematic process would make scientific research more accessible, more repeatable, and could combine the insights of many different minds into one result.

Crowdsourcing the Decomposition Process

There are many other possible creative domains to decompose: writing engaging novels that inspire children to read, authoring instructional materials in math and science that are entertaining and educational, creating diagrams that encapsulate an idea visually, designing products that people love, and finding creative solutions to civic and environmental problems. Currently, the process we used to decompose humor into microtasks is time consuming and it could take years to decompose each new creative field. To more easily extended the HumorTools approach to lots of new domains, we need tools to support the process: mining literature for patterns, analyzing examples, and creating tutorials for each microtask such that the average user can understand the task and perform it well.

The process of analyzing humor to find reusable patterns is essentially sensemaking. The tool we used to support the humor analysis and sensemaking process was Frenzy [**?**]. In the future, we can build on the insights of Cascade [**?**] and Frenzy [**?**] for crowdsourcing data organization to design tools that support distributed sensemaking. The biggest improvement we could make to Frenzy is adding machine learning to suggest patterns or to suggest other items to add to an existing pattern. This would decrease the amount of time spent searching for connections between items, as was evident in the case study presented by Russell et al. [**?**] which cites automated clustering as the main gain of using software systems to support the sensemaking process. By better supporting the sensemaking process, we can speed up the decomposition of many different creative fields into generative processes.

6.4 Summary

Adaptive crowd algorithms can solve large, creative problems that are hard to define and open-ended in their solutions. We believe that adaptive crowd algorithms can solve any problem where the solution space is known to contain design patterns. This includes software engineering, urban planning, graphic design, and screenwriting, to name a few. We believe there are many more problem domains, such as humor, that are not traditionally thought to have design patterns, but where design patterns can be found. This includes writing opinion pieces for a newspaper, constructing a legal case, journalism, marketing, and scientific investigation. By using sensemaking tools like Cascade and Frenzy we can use the crowd to identify design patterns in new domains that we can teach the crowd to apply.

This dissertation has uncovered new approaches and solutions to large, open-ended problems. In the fields of systems and algorithms, adaptive crowd algorithms are a new way of decomposing problems that uses people's conceptual abilities systematically. In the field of crowdsourcing, adaptive crowd algorithms are a new way to combine people's ideas into a cohesive solution using search and design patterns.

BIBLIOGRAPHY

[1] A.M. Turing Award Allen Newell United States - 1975. http://amturing.acm.org/award_winners/newell_3167755.cfm. Accessed: 2016-06-03.

[2] A.M. Turing Award Herbert ("Herb") Alexander Simon United States - 1975. http://amturing.acm.org/award_winners/simon_1031467.cfm. Accessed: 2016-06-03.

[3] Amazon Mechanical Turk. https://www.mturk.com/mturk/welcome. Accessed: 2015-12-01.

[4] CastingWords. https://castingwords.com/. Accessed: 2015-12-01.

[5] Galaxy Zoo. https://www.galaxyzoo.org/. Accessed: 2016-06-02.

[6] The Onion. http://www.theonion.com/. Accessed: 2015-12-01.

[7] The Onion American Voices. http://www.theonion.com/features/american-voices. Accessed: 2015-12-01.

[8] Quirky. https://www.quirky.com/shop. Accessed: 2015-12-01.

[9] Stanford Encyclopedia of Philosophy, Embodied Cognition. http://plato.stanford.edu/entries/embodied-cognition/. Accessed: 2015-12-01.

[10] Stanford Encyclopedia of Philosophy, Philosophy of Humor. http://plato.stanford.edu/entries/humor/. Accessed: 2015-12-01.

[11] Wikipedia. https://en.wikipedia.org/. Accessed: 2016-06-03.

[12] Wikipedia Portal. https://en.wikipedia.org/wiki/Portal:Contents/Portals. Accessed: 2016-06-03.

[13] Maneesh Agrawala, Wilmot Li, and Floraine Berthouzoz. Design principles for visual communication. *Commun. ACM*, 54(4):60–69, April 2011.

[14] Maneesh Agrawala, Doantam Phan, Julie Heiser, John Haymaker, Jeff Klingner, Pat Hanrahan, and Barbara Tversky. Designing effective step-by-step assembly instructions. *ACM Trans. Graph.*, 22(3):828–837, July 2003.

[15] Maneesh Agrawala and Chris Stolte. Rendering effective route maps: Improving usability through generalization. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '01, pages 241–249, New York, NY, USA, 2001. ACM.

[16] Luis Von Ahn, Manuel Blum, Nicholas J. Hopper, and John Langford. Captcha: Using hard ai problems for security. In *Proceedings of the 22Nd International Conference on Theory and Applications of Cryptographic Techniques*, EUROCRYPT'03, pages 294–311, Berlin, Heidelberg, 2003. Springer-Verlag.

[17] Christopher Alexander, Sara Ishikawa, Murray Silverstein, Max Jacobson, Ingrid Fiksdahl-King, and Shlomo Angel. *A Pattern Language - Towns, Buildings, Construction*. Oxford University Press, 1977.

[18] Paul Andre, Haoqi Zhang, Juho Kim, Lydia B. Chilton, Steven P. Dow, and Robert C. Miller. Community clustering: Leveraging an academic crowd to form coherent conference sessions. In Bjorn Hartman and Eric Horvitz, editors, *HCOMP*. AAAI, 2013.

[19] Michael S. Bernstein, Greg Little, Robert C. Miller, Björn Hartmann, Mark S. Ackerman, David R. Karger, David Crowell, and Katrina Panovich. Soylent: A word processor with a crowd inside. In *Proceedings of the 23Nd Annual ACM Symposium on User Interface Software and Technology*, UIST '10, pages 313–322, New York, NY, USA, 2010. ACM.

[20] Jeffrey P. Bigham, Chandrika Jayant, Hanjie Ji, Greg Little, Andrew Miller, Robert C. Miller, Robin Miller, Aubrey Tatarowicz, Brandyn White, Samual White, and Tom Yeh. Vizwiz: Nearly real-time answers to visual questions. In *Proceedings of the 23Nd Annual ACM Symposium on User Interface Software and Technology*, UIST '10, pages 333–342, New York, NY, USA, 2010. ACM.

[21] Kim Binsted and Graeme Ritchie. Computational rules for generating punning riddles. humor. *International Journal of Humor Research*, 10(1):25–76, 1997.

[22] David M. Blei, Andrew Y. Ng, and Michael I. Jordan. Latent dirichlet allocation. *Journal of Machine Learning Research*, 3:993–1022, March 2003.

[23] Jonathan Bragg, Mausam, and Daniel S. Weld. Crowdsourcing multi-label classification for taxonomy creation. In *Proceedings of the First AAAI Conference on Human Computation and Crowdsourcing, HCOMP 2013, November 7-9, 2013, Palm Springs, CA, USA*, 2013.

[24] Richard Buchanan. Wicked problems in design thinking. *Design Issues*, 8(2), 1992.

[25] Bill Buxton. *Sketching User Experiences: Getting the Design Right and the Right Design*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2007.

[26] Judy Carter. *The Comedy Bible: From Stand-up to Sitcom–The Comedy Writer's Ultimate "How To" Guide*. Touchstone, New York, New York, USA, 2001.

[27] Lydia B. Chilton, Juho Kim, Paul André, Felicia Cordeiro, James A. Landay, Daniel S. Weld, Steven P. Dow, Robert C. Miller, and Haoqi Zhang. Frenzy: Collaborative data organization for creating conference sessions. In *Proceedings of the 32Nd Annual ACM Conference on Human Factors in Computing Systems*, CHI '14, pages 1255–1264, New York, NY, USA, 2014. ACM.

[28] Lydia B. Chilton, Greg Little, Darren Edge, Daniel S. Weld, and James A. Landay. Cascade: crowdsourcing taxonomy creation. In *2013 ACM SIGCHI Conference on Human Factors in Computing Systems, CHI '13, Paris, France, April 27 - May 2, 2013*, pages 1999–2008, 2013.

[29] 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.

[30] Peng Dai, Mausam, and Daniel S. Weld. Decision-theoretic control of crowd-sourced workflows. In *In the 24th AAAI Conference on Artificial Intelligence (AAAI10*, 2010.

[31] Walter S. Lasecki Dan Liebling, Jaime Teevan. Selfsourcing personal tasks. ACM, April 2014.

[32] Dmitry Davidov, Oren Tsur, and Ari Rappoport. Semi-supervised recognition of sarcastic sentences in twitter and amazon. In *Proceedings of the Fourteenth Conference on Computational Natural Language Learning*, CoNLL '10, pages 107–116, Stroudsburg, PA, USA, 2010. Association for Computational Linguistics.

[33] Greg Dean. *Step by Step to Stand-Up Comedy*. Heinemann Drama, Portsmouth, New Hampshire, 2000.

[34] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: Simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113, January 2008.

[35] Kees Dorst and Judith Dijkhuis. Analysing design activity comparing paradigms for describing design activity. *Design Studies*, 16(2):261 – 274, 1995.

[36] Douglas K. Van Duyne, James Landay, and Jason I. Hong. *The Design of Sites: Patterns, Principles, and Processes for Crafting a Customer-Centered Web Experience*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2002.

[37] Erich Gamma, Richard Helm, Ralph E. Johnson, and John Vlissides. *Design patterns: elements of reusable object-oriented software*, volume 206. 1995.

[38] Ryan G. Gomes, Peter Welinder, Andreas Krause, and Pietro Perona. Crowdclustering. In J. Shawe-Taylor, R.S. Zemel, P.L. Bartlett, F. Pereira, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems 24*, pages 558–566. Curran Associates, Inc., 2011.

[39] David Alan Grier. The math tables project of the work projects administration: The reluctant start of the computing era. *IEEE Ann. Hist. Comput.*, 20(3):33–50, July 1998.

[40] Jonathan Grudin. Groupware and social dynamics: Eight challenges for developers. *Commun. ACM*, 37(1):92–105, January 1994.

[41] Jonathan Guryan, Kory Kroft, and Matthew J. Notowidigdo. Peer effects in the workplace: Evidence from random groupings in professional golf tournaments. *American Economic Journal: Applied Economics*, 1(4):34–68, October 2009.

[42] Bjorn Hartmann. *Gaining Design Insight Through Interaction Prototyping Tools*. PhD thesis, Stanford University, 2009.

[43] Björn Hartmann, Loren Yu, Abel Allison, Yeonsoo Yang, and Scott R. Klemmer. Design as exploration: Creating interface alternatives through parallel authoring and runtime tuning. In *Proceedings of the 21st Annual ACM Symposium on User Interface Software and Technology*, UIST '08, pages 91–100, New York, NY, USA, 2008. ACM.

[44] Kurtis Heimerl, Brian Gawalt, Kuang Chen, Tapan Parikh, and Björn Hartmann. Communitysourcing: Engaging local crowds to perform expert work via physical kiosks. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '12, pages 1539–1548, New York, NY, USA, 2012. ACM.

[45] Sally Holloway. *The Serious Guide to Joke Writing: How To Say Something Funny About Anything*. Bookshaker, Great Yarmouth, UK, 2010.

[46] Jeff Howe. The rise of crowdsourcing. *Wired Magazine*, 14(6), 06 2006.

[47] William Hudson. *Card Sorting In "The Encyclopedia of Human-Computer Interaction, 2nd Ed.".* 2012.

[48] Panagiotis G. Ipeirotis, Foster Provost, and Jing Wang. Quality management on amazon mechanical turk. In *Proceedings of the ACM SIGKDD Workshop on Human Computation*, HCOMP '10, pages 64–67, New York, NY, USA, 2010. ACM.

[49] J. Joyce. *Ulysses*. Number v. 1. Odyssey Press, 1939.

[50] Steve Kaplan. *The Hidden Tools of Comedy: The Serious Business of Being Funny*. Michael Wiese Productions, Studio City, CA, 2013.

[51] Dimitris Karampinas and Peter Triantafillou. Crowdsourcing taxonomies. In Elena Simperl, Philipp Cimiano, Axel Polleres, Oscar Corcho, and Valentina Presutti, editors, *The Semantic Web: Research and Applications*, volume 7295 of *Lecture Notes in Computer Science*, pages 545–559. Springer Berlin Heidelberg, 2012.

[52] Chloé Kiddon and Yuriy Brun. That's what she said: Double entendre identification. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies: Short Papers - Volume 2*, HLT '11, pages 89–94, Stroudsburg, PA, USA, 2011. Association for Computational Linguistics.

[53] Joy Kim, Mira Dontcheva, Wilmot Li, Michael S. Bernstein, and Daniela Steinsapir. Motif: Supporting novice creativity through expert patterns. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, CHI '15, pages 1211–1220, New York, NY, USA, 2015. ACM.

[54] Juho Kim, Phu Tran Nguyen, Sarah Weir, Philip J. Guo, Robert C. Miller, and Krzysztof Z. Gajos. Crowdsourcing step-by-step information extraction to enhance existing how-to videos. In *Proceedings of the 32Nd Annual ACM Conference on Human Factors in Computing Systems*, CHI '14, pages 4017–4026, New York, NY, USA, 2014. ACM.

[55] Juho Kim, Haoqi Zhang, Paul André, Lydia B. Chilton, Wendy Mackay, Michel Beaudouin-Lafon, Robert C. Miller, and Steven P. Dow. Cobi: A community-informed conference scheduling tool. In *Proceedings of the 26th Annual ACM Symposium on User Interface Software and Technology*, UIST '13, pages 173–182, New York, NY, USA, 2013. ACM.

[56] Aniket Kittur, Boris Smus, Susheel Khamkar, and Robert E. Kraut. Crowdforge: Crowdsourcing complex work. In *Proceedings of the 24th Annual ACM Symposium on User Interface Software and Technology*, UIST '11, pages 43–52, New York, NY, USA, 2011. ACM.

[57] Travis Kriplean, Jonathan Morgan, Deen Freelon, Alan Borning, and Lance Bennett. Supporting reflective public thought with considerit. In *Proceedings of the ACM 2012 Conference on Computer Supported Cooperative Work*, CSCW '12, pages 265–274, New York, NY, USA, 2012. ACM.

[58] Anand Kulkarni, Matthew Can, and Björn Hartmann. Collaboratively crowdsourcing workflows with turkomatic. In *Proceedings of the ACM 2012 Conference on Computer Supported Cooperative Work*, CSCW '12, pages 1003–1012, New York, NY, USA, 2012. ACM.

[59] James A. Landay and Brad A. Myers. Interactive sketching for the early stages of user interface design. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '95, pages 43–50, New York, NY, USA, 1995. ACM Press/Addison-Wesley Publishing Co.

[60] David Laniado, Davide Eynard, and Marco Colombetti. Using wordnet to turn a folksonomy into a hierarchy of concepts. In *Semantic Web Application and Perspectives - Fourth Italian Semantic Web Workshop*, pages 192–201, dec 2007.

[61] Walter Lasecki, Christopher Miller, Adam Sadilek, Andrew Abumoussa, Donato Borrello, Raja Kushalnagar, and Jeffrey Bigham. Real-time captioning by groups of non-experts. In *Proceedings of the 25th Annual ACM Symposium on User Interface Software and Technology*, UIST '12, pages 23–34, New York, NY, USA, 2012. ACM.

[62] Walter S. Lasecki, Kyle I. Murray, Samuel White, Robert C. Miller, and Jeffrey P. Bigham. Real-time crowd control of existing interfaces. In *Proceedings of the 24th Annual ACM Symposium on User Interface Software and Technology*, UIST '11, pages 23–32, New York, NY, USA, 2011. ACM.

[63] Walter S. Lasecki, Rachel Wesley, Jeffrey Nichols, Anand Kulkarni, James F. Allen, and Jeffrey P. Bigham. Chorus: A crowd-powered conversational assistant. In *Proceedings of the 26th Annual ACM Symposium on User Interface Software and Technology*, UIST '13, pages 151–162, New York, NY, USA, 2013. ACM.

[64] E. Law, B. Settles, A. Snook, H. Surana, L. von Ahn, and T. Mitchell. Human computation for attribute and attribute value acquisition. In *Proceedings of the CVPR Workshop on Fine-Grained Visual Categorization*, 2011.

[65] Edith Law and Luis von Ahn. Input-agreement: A new mechanism for collecting data using human computation games. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '09, pages 1197–1206, New York, NY, USA, 2009. ACM.

[66] Edith Law, Luis von Ahn, and Tom Mitchell. Search war: A game for improving web search. In *Proceedings of the ACM SIGKDD Workshop on Human Computation*, HCOMP '09, pages 31–31, New York, NY, USA, 2009. ACM.

[67] Jeehyung Lee, Wipapat Kladwang, Minjae Lee, Daniel Cantu, Martin Azizyan, Hanjoo Kim, Alex Limpaecher, Snehal Gaikwad, Sungroh Yoon, Adrien Treuille, Rhiju Das, and EteRNA Participants. Rna design rules from a massive open laboratory. *Proceedings of the National Academy of Sciences*, 111(6):2122–2127, 2014.

[68] Greg Little, Lydia B. Chilton, Max Goldman, and Robert C. Miller. TurKit: Tools for Iterative Tasks on Mechanical Turk. In *Human Computation Workshop (HComp2009)*, 2009.

[69] Greg Little, Lydia B. Chilton, Max Goldman, and Robert C. Miller. Turkit: Human computation algorithms on mechanical turk. In *Proceedings of the 23Nd Annual ACM Symposium on User Interface Software and Technology*, UIST '10, pages 57–66, New York, NY, USA, 2010. ACM.

[70] Kurt Luther, Amy Pavel, Wei Wu, Jari-lee Tolentino, Maneesh Agrawala, Björn Hartmann, and Steven P. Dow. Crowdcrit: Crowdsourcing and aggregating visual design critique. In *Proceedings of the Companion Publication of the 17th ACM Conference on Computer Supported Cooperative Work & Social Computing*, CSCW Companion '14, pages 21–24, New York, NY, USA, 2014. ACM.

[71] Matthew M. Hurley and Daniel C. Dennett. *Inside Jokes: Using Humor to Reverse-Engineer the Mind*. The MIT Press, Cambridge, MA, USA, 2011.

[72] Pamela McCorduck. *Machines Who Think: A Personal Inquiry into the History and Prospects of Artificial Intelligence*. AK Peters Ltd, 2004.

[73] Paul Merrell, Eric Schkufza, Zeyang Li, Maneesh Agrawala, and Vladlen Koltun. Interactive furniture layout using interior design guidelines. *ACM Trans. Graph.*, 30(4):87:1–87:10, July 2011.

[74] Mark W. Newman and James A. Landay. Sitemaps, storyboards, and specifications: A sketch of web site design practice. In *Proceedings of the 3rd Conference on Designing Interactive Systems: Processes, Practices, Methods, and Techniques*, DIS '00, pages 263–274, New York, NY, USA, 2000. ACM.

[75] Mark W. Newman, James Lin, Jason I. Hong, and James A. Landay. Denim: An informal web site design tool inspired by observations of practice. *Hum.-Comput. Interact.*, 18(3):259–324, September 2003.

[76] Donald A. Norman. *The Design of Everyday Things.* Basic Books, Inc., New York, NY, USA, 2002.

[77] Jon Noronha, Eric Hysen, Haoqi Zhang, and Krzysztof Z. Gajos. Platemate: Crowdsourcing nutritional analysis from food photographs. In *Proceedings of the 24th Annual ACM Symposium on User Interface Software and Technology*, UIST '11, pages 1–12, New York, NY, USA, 2011. ACM.

[78] Devi Parikh and Kristen Grauman. Interactively building a discriminative vocabulary of nameable attributes. In *Proceedings of the 2011 IEEE Conference on Computer Vision and Pattern Recognition*, CVPR '11, pages 1681–1688, Washington, DC, USA, 2011. IEEE Computer Society.

[79] Gene Perret. *The Comedy Writing Workbook.* Players Press, 1994.

[80] Daniel H. Pink. *Drive: The Surprising Truth About What Motivates Us.* Riverhead Books, 2011.

[81] Peter Pirolli and Stuart Card. The sensemaking process and leverage points for analyst technology as identified through cognitive task analysis. pages 2–4, 2005.

[82] Alexander J. Quinn and Benjamin B. Bederson. Human computation: A survey and taxonomy of a growing field. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '11, pages 1403–1412, New York, NY, USA, 2011. ACM.

[83] Victor Raskin. *The Semantic Mechanisms of Humor.* Springer Netherlands, 1984.

[84] Victor Raskin. *The Primer of Humor Research.* De Gruyter, Berlin, Germany, 2009.

[85] Daniel M. Russell, Mark J. Stefik, Peter Pirolli, and Stuart K. Card. The cost structure of sensemaking. In *Proceedings of the INTERACT '93 and CHI '93 Conference on Human Factors in Computing Systems*, CHI '93, pages 269–276, New York, NY, USA, 1993. ACM.

[86] D.A. Schon. *The Reflective Practitioner: How Professionals Think In Action.* Basic Books. Basic Books, 1984.

[87] Dafna Shahaf, Eric Horvitz, and Robert Mankoff. Inside jokes: Identifying humorous cartoon captions. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '15, pages 1065–1074, New York, NY, USA, 2015. ACM.

[88] Herbert A. Simon. *The Sciences of the Artificial (3rd Ed.)*. MIT Press, Cambridge, MA, USA, 1996.

[89] Herbert A. Simon and Allan Newell. Human problem solving: The state of the theory in 1970. *American Psychologist*, 26(2), 1971.

[90] Herbert A. Simon, Allan Newell, and J. C. Shaw. Elements of a theory of human problem solving. *Psychological Review*, 65(3), 1958.

[91] Omer Tamuz, Ce Liu, Serge Belongie, Ohad Shamir, and Adam Tauman Kalai. Adaptively learning the crowd kernel. In *In ICML11*, 2011.

[92] Julia M. Taylor and Lawrence J. Mazlack. Computationally recognizing wordplay in jokes. In *In Proceedings of CogSci 2004*, 2004.

[93] Alan M. Turing. On computable numbers, with an application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society*, 2(42):230–265, 1936.

[94] Luis von Ahn and Laura Dabbish. Labeling images with a computer game. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '04, pages 319–326, New York, NY, USA, 2004. ACM.

[95] 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.

[96] John Vorhaus. *The Comic Toolbox: How to Be Funny Even If You're Not*. Silman James Press, Los Angeles, CA, 1994.

[97] Jacob Whitehill, Paul Ruvolo, Tingfan Wu, Jacob Bergsma, and Javier Movellan. Whose vote should count more: Optimal integration of labels from laberlers of unknown expertise. In *In Proc. of NIPS*, pages 2035–2043, 2009.

[98] Anbang Xu, Shih-Wen Huang, and Brian Bailey. Voyant: Generating structured feedback on visual designs using a crowd of non-experts. In *Proceedings of the 17th ACM Conference on Computer Supported Cooperative Work & Social Computing*, CSCW '14, pages 1433–1444, New York, NY, USA, 2014. ACM.

[99] J. Yi, J. Rong, A. Jain, and S. Jain. Crowdclustering with sparse pairwise labels: A matrix completion approach. In *HCOMP 2012*, 2010.

[100] Lixiu Yu, Aniket Kittur, and Robert E. Kraut. Distributed analogical idea generation: Inventing with crowds. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '14, pages 1245–1254, New York, NY, USA, 2014. ACM.

[101] Lixiu Yu, Aniket Kittur, and Robert E. Kraut. Searching for analogical ideas with crowds. In *Proceedings of the 32Nd Annual ACM Conference on Human Factors in Computing Systems*, CHI '14, pages 1225–1234, New York, NY, USA, 2014. ACM.

[102] Lixiu Yu and Jeffrey V. Nickerson. Cooks or cobblers?: Crowd creativity through combination. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '11, pages 1393–1402, New York, NY, USA, 2011. ACM.

[103] Haoqi Zhang, Edith Law, Rob Miller, Krzysztof Gajos, David Parkes, and Eric Horvitz. Human computation tasks with global constraints. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '12, pages 217–226, New York, NY, USA, 2012. ACM.