

Chapter 2

Human Computation Algorithms

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

Unlike employees in traditional firms or dedicated volunteers on Wikipedia who tend to be available over time, keep track of context, and generally provide good solutions, the types of crowds we can readily recruit may consist of individuals who are only briefly involved in any particular task (e.g., up to 10 minutes), and include individuals who may or may not provide helpful inputs. Given these characteristics,

an effective human computation algorithm that calls on such crowds must break down complex problems into smaller tasks, and synthesize noisy inputs from large numbers of individuals to provide quality solutions.

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

2.1 Design Patterns

2.1.1 Design Pattern 1: Divide-and-Conquer

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

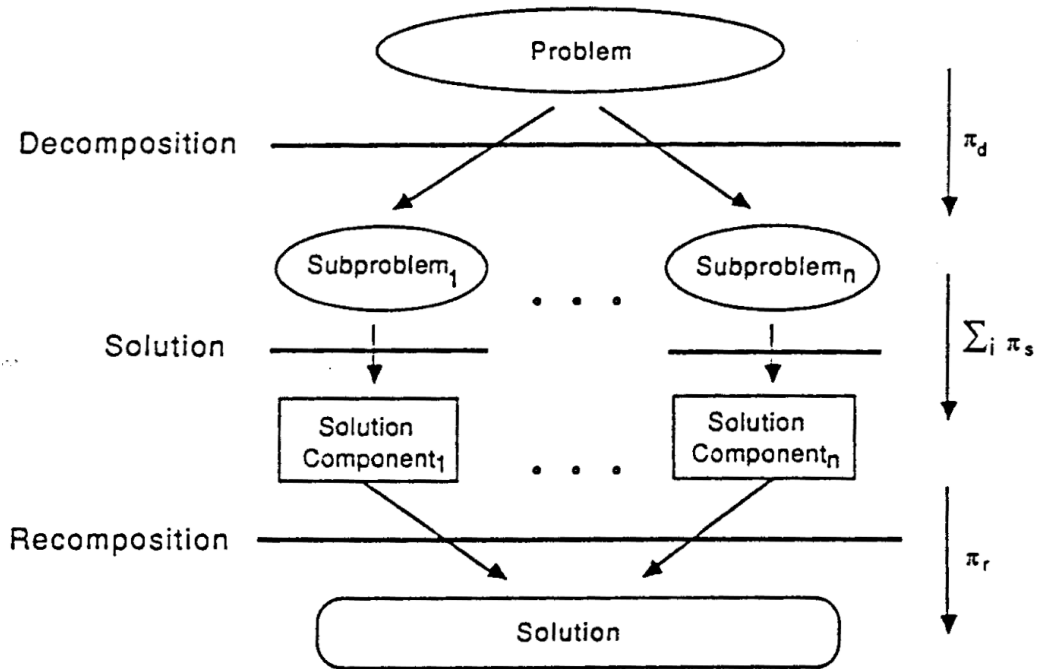


Figure 2.1: Diagram depicting the decompose, solve, and recombine structure of divide-and-conquer algorithms, with reasoning about costs and tradeoffs associated with different phases of the problem solving process. From Horvitz [34].

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

2.1.2 Design Pattern 2: Redundancy-based Quality Control

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

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

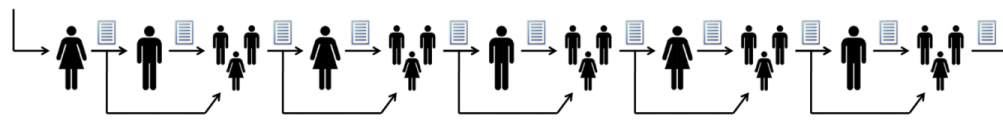
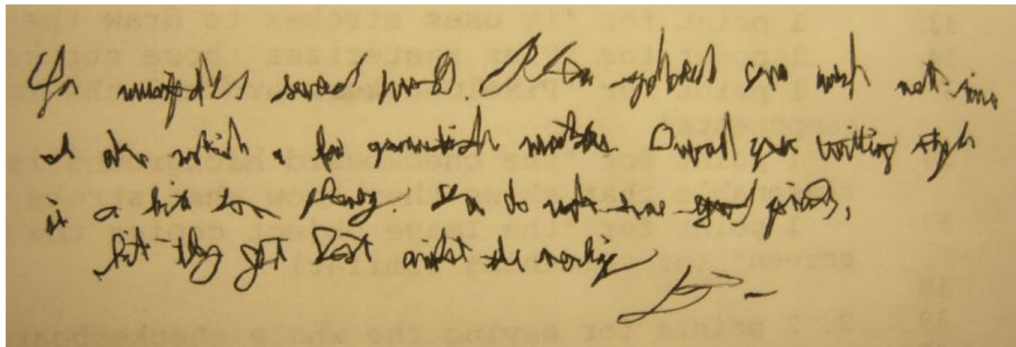
While it is important to be able to determine the correct answer from a set of collected answers, it is just as important to collect good answers in the first place. In addition to providing workers with proper instructions, examples, and training, requesters can attempt to elicit good faith efforts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

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

2.1.3 Design Pattern 3: Iterative Improvement

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

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



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

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

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

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

2.2 Case Study: Audio Transcription

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

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

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

criptions, grade transcriptions, and improve transcriptions.¹ CastingWords charges between \$1 and \$2.50 per minute of audio transcribed depending on the required turnaround time, and pays workers based on the quality of the transcription and the task's difficulty.

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

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

2.2.1 Designing an Algorithm

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

¹<http://castingwords.com/>

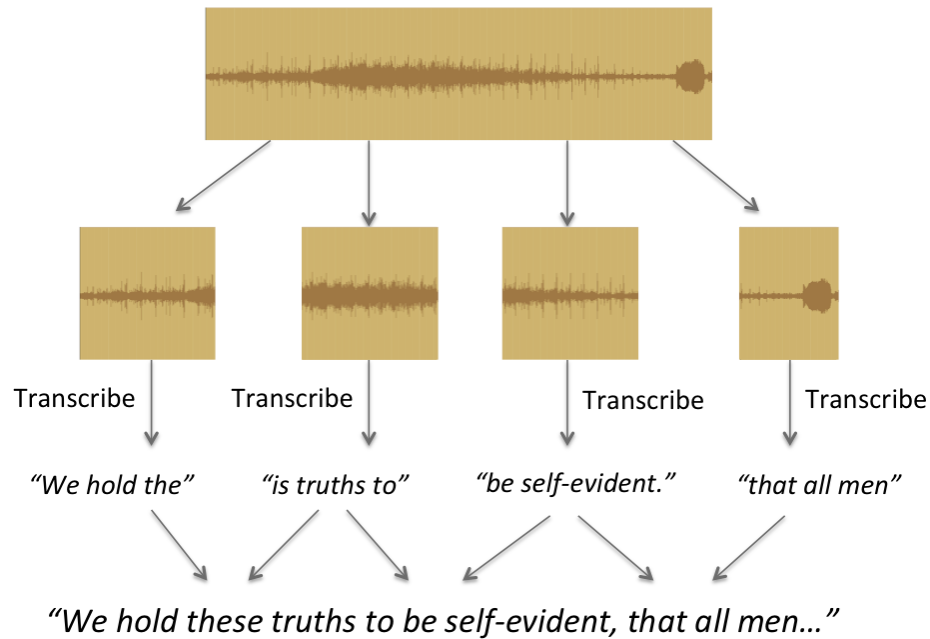


Figure 2.3: Divide-and-conquer is applied to crowdsourcing audio transcription. An audio recording is partitioned into shorter clips which are transcribed and then rejoined.

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

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

To rejoin transcripts of adjacent clips, we need to address the possibility that a word may have been split when the clip was initially divided. One solution is to

combine two adjacent audio clips and present it along with the transcriptions of the adjacent audio clips for a contributor to join. To ensure that the person only edits where the two transcripts join, we can allow the person to only edit the transcript near the middle of a combined transcript. As rejoining transcripts can be a difficult task for anyone to complete correctly, we can again apply the iterative design pattern for this recompose step.

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

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

To address these questions, let us take a step back from the iterative design pattern and think about what happens if we ask two people to transcribe the same audio clip

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

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

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

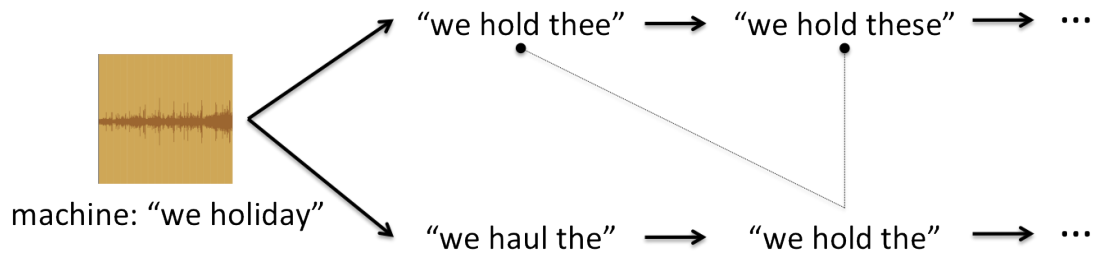


Figure 2.4: Contributors are alternately assigned to one of two pathways. They modify previous transcripts from their own pathway, and their transcripts are scored based on how well they match recent entries in the opposite pathway.

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

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

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

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

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

2.2.2 Experiments

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



Figure 2.5: A screenshot of the user interface

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

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

Figure 2.5 provides a screenshot of the user interface for the iterative dual pathway version of the game. In this version, players transcribed each clip and were awarded

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

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

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

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

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

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

on a character basis as follows:

$$\text{LD} = \text{Insertions} + \text{Deletions} + \text{Substitutions} \quad (2.2)$$

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

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

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

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

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

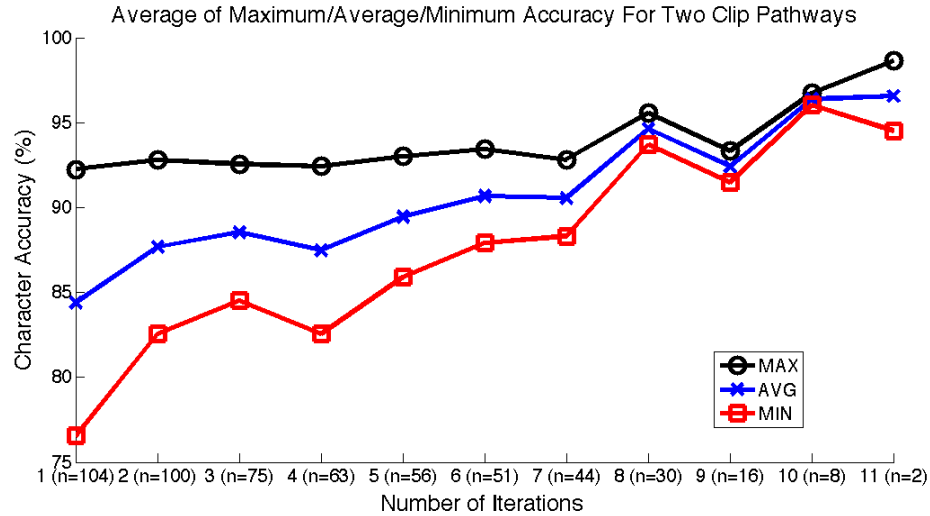


Figure 2.6: The average maximum/average/minimum accuracies of transcripts in the two pathways after k iterations. Transcripts are removed from this graph after they converge to avoid an upward bias.

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

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

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

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

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

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

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

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

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

(“Dear God” should be “If you go”)

Players were also adept at rejoining clips:

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

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

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

(Correct)

2.2.3 Summary

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

2.3 Case Study: Nutrition Analysis

In the audio transcription algorithm, human effort is elicited only for providing transcripts. In general, human computation algorithms can harness different types of human effort and coordinate the inputs and outputs of different tasks. For some problems, the algorithmic challenge is identifying what types of effort to elicit, and effectively coordinating heterogeneous contributions to derive a solution. In this sec-

tion, we demonstrate how to combine our understanding of how an expert performs a task with our understanding of the crowd to derive an effective solution for crowd-sourced nutrition analysis.

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

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

RFPM thus combines the accuracy of direct observation by experts with the convenience of free-living conditions. Users of the method found it satisfying and easy to use [65]. The problem is cost and scarcity. RFPM relies on experts to analyze each photograph, limiting the system's accessibility and potential scale. The method

might be feasible in specific healthcare settings, but trained dietitians are too costly and scarce for general use.

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

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

2.3.1 PlateMate

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

³<http://mealsnap.com/>, accessed July 5, 2011

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

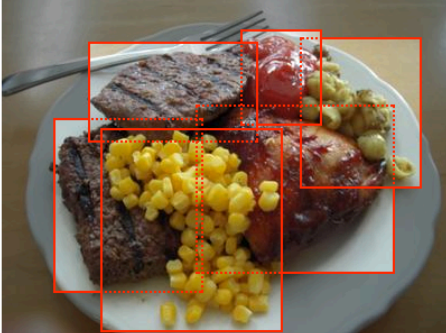


	kcal	fat (g)	carbs (g)	protein (g)
DINNER	1573.2	72.9	84	138.9
				
Yellow Corn (0.50 cup)	303	3.9	61.6	7.8
barbeque chicken breast				
Chicken Breast Meat and Skin (Broilers or Fryers) (1.00 breast, bone removed)	249	13.4	0	30.2
Barbeque Sauce (Low Sodium, Canned) (0.14 cup)	26.6	0.6	4.5	0.6
Beef Steak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

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

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

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

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

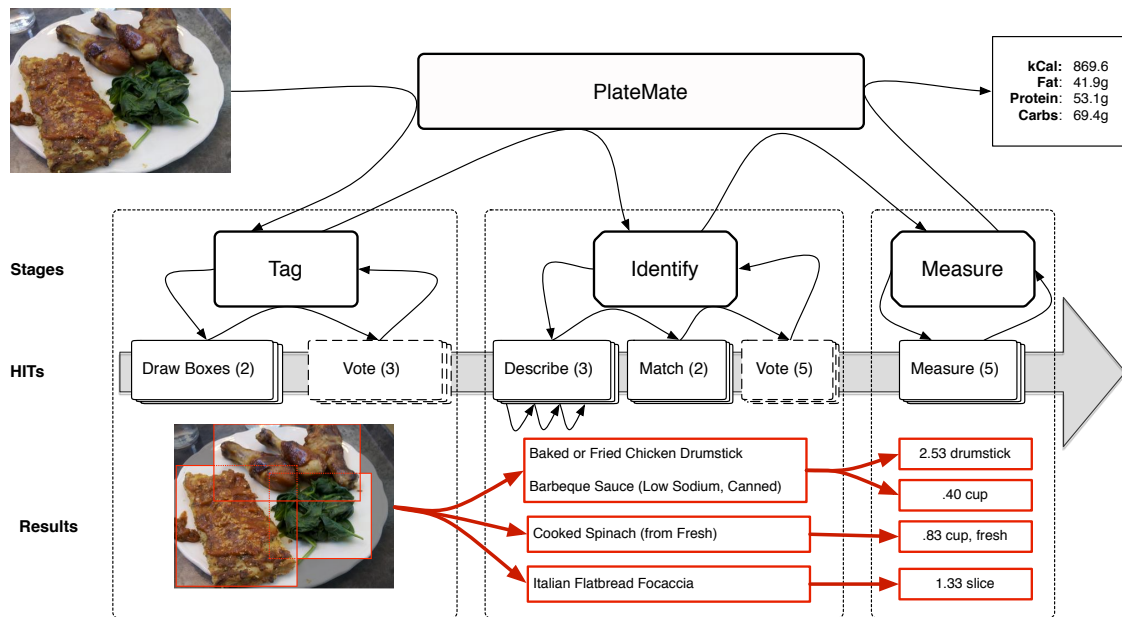


Figure 2.8: The PlateMate system. Work travels between stages and Human Intelligence Tasks (HITs) along the black arrows, starting from the input on the left and concluding with the output on the right. The system takes submitted photos and creates Tag tasks to annotate these photos with boxes. Each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

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

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

Step 1: Tag

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

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

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

Similarity Comparison and Voting: Two Turkers are asked to tag each photo. Once both assignments are completed, they are algorithmically compared based on the number, size, and position of boxes. If the two groups are sufficiently similar,

one is picked at random as the final answer. If the box groups differ significantly, three additional Turkers are shown each set overlaid on the photo and asked to select the better option, using similar guidelines. The box group receiving more votes is returned as the final result.

Step 2: Identify

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

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

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

These problems may also have occurred because the interface asked Turkers to perform two conceptually different tasks sequentially but only produce one final output. Turkers first had to identify each food and then locate the corresponding entry

in the database. To correct for this, we developed a workflow that contained two simpler HITs. The first asks workers to describe the food in their own words. The second asks (other) workers to match this description to items in the database.

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

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

Matching Foods: After three iterations, the output of the Describe task is fed into a Match HIT. Here, workers see the photo (with the box) and the final descriptions. They are asked to select the best entry or set of entries in the database to match the boxed portion of the photo, with the descriptions serving as suggestions for what to search. Workers first attempt to locate the description of the box as a whole in the

database. If they do not find a good match, they search for each part. For example, workers should first search for “salad with chicken and tomatoes.” If this fails, they should look for “chicken breast,” “romaine lettuce,” and “cherry tomatoes.”

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

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

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

Step 3: Measure

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

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

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

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

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

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

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

Turker Qualifications

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

2.3.2 Evaluation

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

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

Method

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



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

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

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

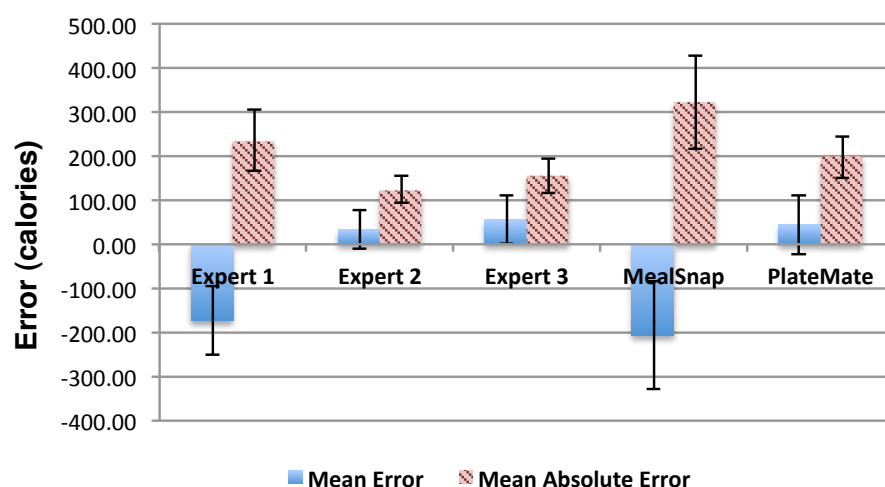


Figure 2.10: Mean errors (i.e., overall bias) and mean absolute errors (average magnitude of an error) for estimates made by the human experts, the Meal Snap application, and PlateMate compared to data provided by manufacturer or preparer. Error bars represent standard errors.

Results

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

Absolute error rates reflect the average magnitude of the error, but not the biases in each method. To understand how estimates from each source would add up over

time, we also measured mean error without taking absolute values. The best expert overestimated by just 32.75 calories on average, for a mean error rate of +5.5%. The other two experts had error rates of +9.2% and -27.5%.

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

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

Error Analysis

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

Most errors occurred in the Identify stage. Turkers had trouble distinguishing similar types of a food, which sometimes had large nutrition differences. A plate

of vegetarian baked beans was identified as regular baked beans, tripling the calorie count. Branded foods also caused problems: a relatively low-calorie chicken sandwich was identified as a sandwich from the restaurant Chili's, which had over twice as many calories.

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

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

2.3.3 The Management Framework

Because PlateMate relies primarily on dividing human work into a number of heterogeneous and interacting tasks, and because the issues of worker skill and motivation were central to our design process, we found it conceptually helpful to use

human organizational hierarchies as a metaphor for designing our system. Specifically, we observe that in the real world, expert-level work can sometimes be reproduced by less skilled workers—each working on a specific part of the process—supervised by managers who are not necessarily skilled craftsmen themselves, but who know how to assign tasks, route work among workers, and verify the quality of the work.

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

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

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

and sent up the hierarchy as Tag’s completed output. All managers work in parallel, each processing its own stream of work.

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

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

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

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

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

2.3.4 Summary

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

2.4 Discussion

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

Identify what needs to be done

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

Apply design patterns to overcome crowd limitations

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

and combined the iterative design pattern with the output-agreement design pattern to encourage participants to provide accurate improvements.

Design tasks based on the way the crowd works

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

Prototype early and run pilot experiments

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