Chapter 4

Harnessing Crowd Abilities: Control and Synthesis

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

From a computational perspective, we envision that individuals in a crowd can play diverse roles in an organized problem-solving process. People can not only serve as data oracles at the endpoints of computation, but also as modules for decomposing problems, controlling the algorithmic progression, and even generating plans and

synthesizing programs for solving problems. From an organizational perspective, individuals in the crowd may take on roles beyond "doing the work"—including defining and communicating subgoals, evaluating the value of current solutions, and routing tasks to appropriate individuals. The crowd may also be made aware of time or other resource constraints, and be asked to make tradeoffs between further deliberation versus taking time-critical actions.

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

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

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

4.1 Related Work

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

Analogous to our study of general problem solving with crowds, the field of artificial intelligence also concerns itself with general problem solving, but from the perspective of machine agents. Studies of metareasoning [36, 80] aim to design agents

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

4.2 Crowd as Controllers

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

- **Decompose versus solve**

  In Chapter 2, we introduced divide-and-conquer as a useful design pattern for decomposing a problem into subproblems, and for composing solutions of subproblems into a solution. For open-ended tasks in which the crowd performs the decomposition, the difficulty of resulting subtasks may be hard to determine a priori. Instead of predetermining how much a problem should be decomposed before requesting a solution to a subproblem, it may be helpful to give the crowd the option to either solve a problem completely, or to decompose the problem

for the crowd to then solve or decompose further. The crowd's decisions would make implicit tradeoffs between the costs of different stages of computation, potentially enabling more efficient problem solving while also allowing individuals to make decisions based on how much effort they are willing and able to contribute. For example, Zhang et al. [105] introduced a system called TurkSort, which crowdsourced tasks from a quicksort algorithm to Mechanical Turk workers who contributed by finding pivots, partitioning, or sorting, at their choosing. By giving workers the choice of sorting the current list or decomposing the list further, the base case of the recursion was defined implicitly by workers' decisions. As another example, when synthesizing a workflow for solving a problem, Turkomatic [50] workers were asked to judge whether the current price for a task is fair, and if not to decompose it into simpler tasks, with this process repeated recursively.

- **Transmitting solution context and subgoals**

  As part of problem solving, some computational methods track and pass parameters on local and global states and on measures of progress. Human computation may face similar challenges with sharing context among workers about problem-solving strategy and state, particularly when the computation is divided into small pieces performed by many workers. Unless a decomposition is defined or context about what work remains is shared, it may be hard for people to contribute effectively. For example, in the Mobi experiment in Section 3.3, we showed that the absence of todo items significantly increased the amount of time taken to resolve constraints. While we can sometimes rely on the system to

provide the necessary context, we can also engage the crowd in sharing solution context and subgoals. Such actions may enable more efficient problem solving, by helping subsequent contributors to make better decisions and allowing good problem-solving strategies to be passed forward.

- **Controlling search processes**

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

4.2.1 Case Study: 8-Puzzle

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

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

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

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

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

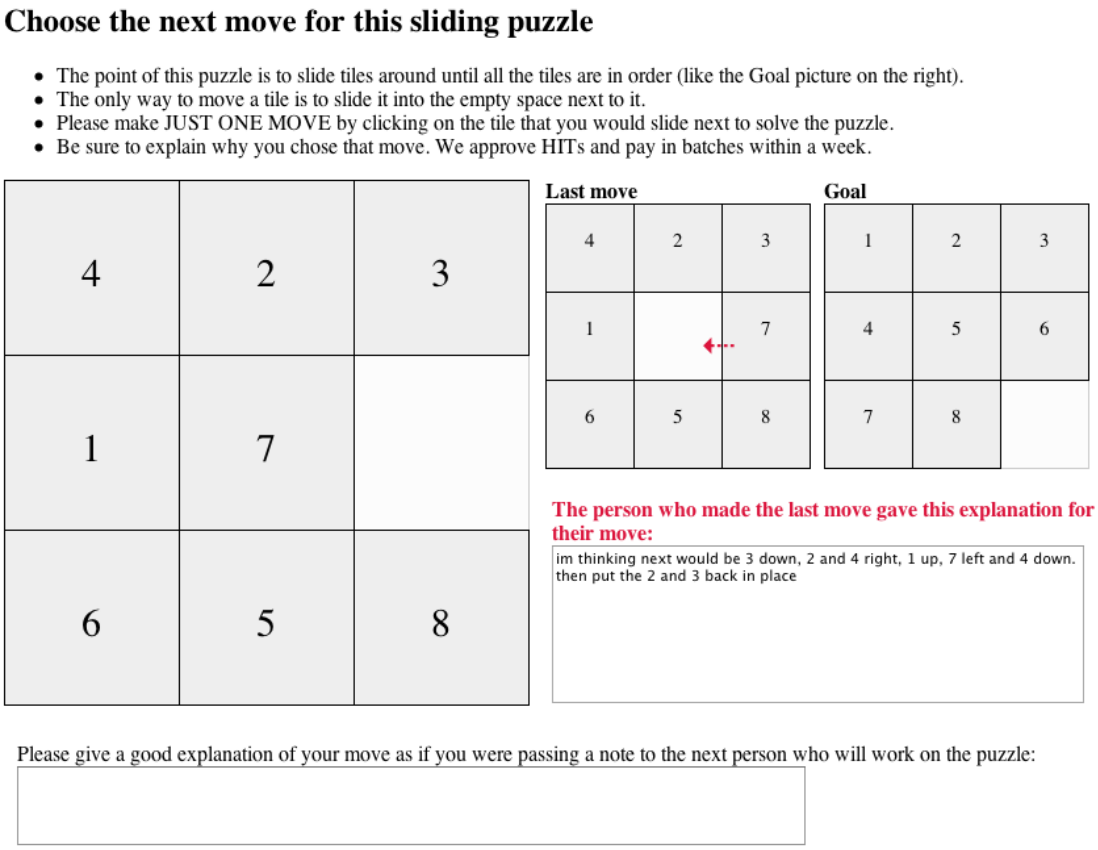


Figure 4.1: Task interface for an iterative step in the 8-puzzle game, where each Turker is shown the last Turker's move and explanation for that move. Here the previous Turker moved the 7 tile and recommended the next sequence of moves.

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

Our results show that in the condition with context passing, all puzzles were solved before 100 steps were reached. In the condition without context passing, 9 of the 10 puzzles were solved for medium difficulty puzzles, and only 5 of the 10 puzzles were solved for hard difficulty puzzles. In addition to completing more puzzles, con-

text passing also reduced the number of iterations Turkers took to complete puzzles. Considering all instances, a Wilcoxon test shows a significant difference (z = -2.61, $p < 0.01$) between the number of steps before a puzzle is solved (or stopped after 100 steps) in the context passing condition ($\mu = 38.6$) and the no context passing baseline ($\mu = 55.9$).

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

4.3 Towards Human Program Synthesis

As the crowd engages in algorithmic control, humans are no longer limited to providing outputs for predefined modules, but can fill in parameters of the algorithm itself and make evaluative decisions to define the best path through a solution space. An interesting question is whether a crowd can go beyond algorithm control towards the notion of *synthesis*. In machine computation, *program synthesis* considers the use of appropriate design tactics to systematically derive a program based on a problem specification. For example, the synthesis of a divide-and-conquer algorithm [86] may

involve the derivation of a tree of specifications, where leaves in the tree represent subproblems for which solutions can be readily provided, and instructions for recomposition are also derived. Taking the analogy to the crowd, we can seek to enlist a crowd in both program synthesis and program execution. By considering problems that the crowd is well-suited for, we can engage the crowd to construct an overall plan for the problem-solving process and to execute the plan. Such plans can include decomposing a problem into subproblems, solving the subproblems, and then recomposing solutions to subproblems into a solution.

4.3.1 Case Study: Collaborative Planning for Web Search

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

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

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

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

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

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

CrowdPlan

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

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

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

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

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

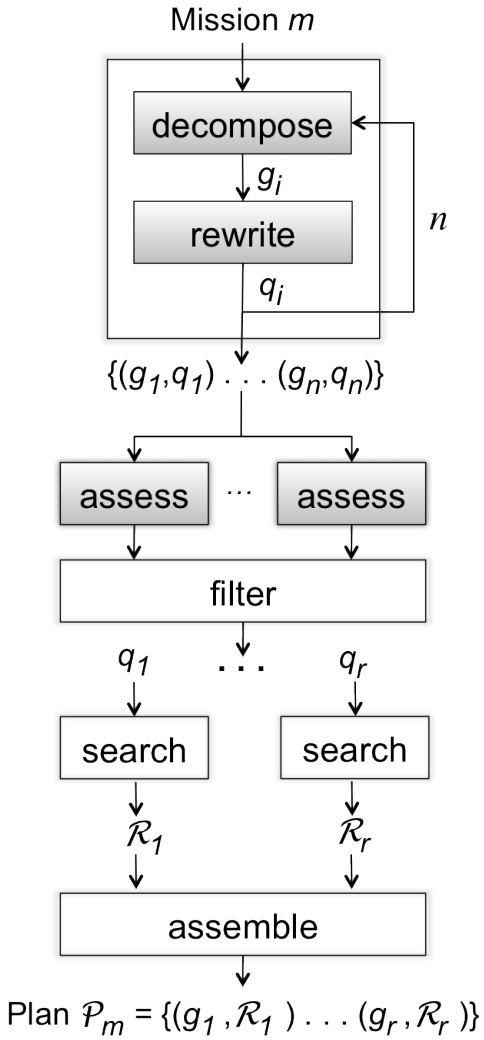


Figure 4.2: CrowdPlan algorithm

- **search**: given a retained search query q_j, this operation retrieves a set of search results $\mathcal{R}_j$ associated with the query.

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

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

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

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

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

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

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

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

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

Figure 4.3: Mission statements submitted by subjects

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

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

Evaluation

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

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

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

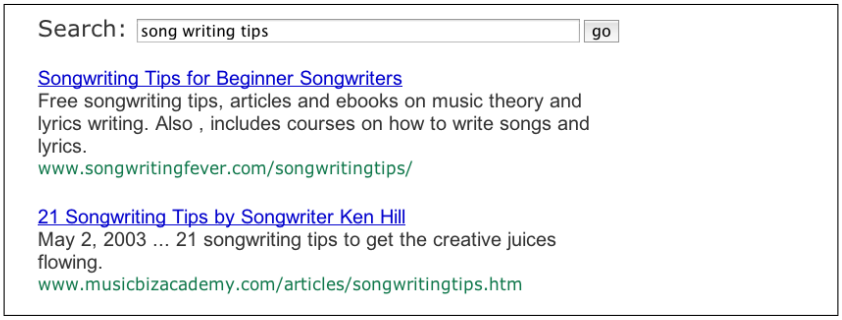
given explanations ($\mu = 9.7$, $\sigma = 2.25$) and the no explanations ($\mu = 8.72$, $\sigma = 2.24$) conditions.

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

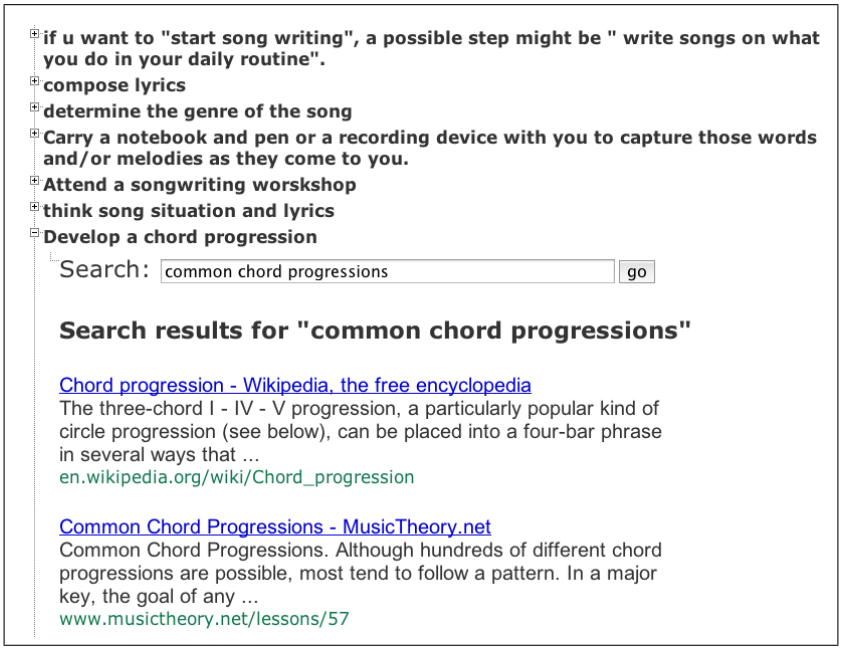
In light of this observation, we created a list-view visualization of the simple plan (see Figure 4.4(b)), which displays the decomposed goals for the mission, the search query associated with each goal, and a short list of five search results. To evaluate the effectiveness of this interface, we asked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

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

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



(a) Standard search



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

Figure 4.4: Standard Search versus Simple Plan

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

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

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

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

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

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

4.4 Discussion

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

We find that in both worker-worker and worker-requester interactions, being able to effectively share and present problem-solving context is crucial. In the 8-puzzle, we observed that short messages about problem-solving strategies were easy to process and of high value when good paths were identified. We saw examples of effective reuse when messages were edited and passed on, and also examples in which workers

identified new strategies when they found suggestions unhelpful. For workers to make such evaluative decisions and act effectively, the problem-solving context provided by workers and the system needs to be easily understandable.

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

As we move toward crowdsourcing general computation, the notion of expertise becomes more prominent as the roles people play become more diverse and specialized. The ability to identify expertise and reward individuals for providing meta-expertise (e.g., controlling the algorithmic process, routing to others who are experts), may

allow us to solve problems that we otherwise would not be able to solve with a crowd. The next chapter introduces methods for *task routing*, that aim to harness the ability of people to both contribute to a solution and guide the problem-solving process by routing tasks to others who they believe can effectively solve and route.