CrowdPlanr: Planning Made Easy with Crowd

Ilia Lotosh Tova Milo Slava Novgorodov

School of Computer Science
Tel-Aviv University
{ilialoto,milo,slavanov}@post.tau.ac.il

***Abstract*—Recent research has shown that crowd sourcing can be used effectively to solve problems that are difficult for computers, e.g., optical character recognition and identification of the structural configuration of natural proteins [1]. In this demo we propose to use the power of the crowd to address yet another difficult problem that frequently occurs in a daily life - planning a sequence of actions, when the goal is hard to formalize. For example, planning the sequence of places/attractions to visit in the course of a vacation, where the goal is to enjoy the resulting vacation the most, or planning the sequence of courses to take in an academic schedule planning, where the goal is to obtain solid knowledge of a given subject domain. Such goals may be easily understandable by humans, but hard or even impossible to formalize for a computer.**

We present a novel algorithm for efficiently harnessing the crowd to assist in solving such planning problems. The algorithm builds the desired plans incrementally, optimally choosing at each step the 'best' questions so that the overall number of questions that need to be asked is minimized. We demonstrate the effectiveness of our solution in CrowdPlanr , a system for vacation travel planning. Given a destination, dates, preferred activities and other constraints CrowdPlanr employs the crowd to build a vacation plan (sequence of places to visit) that is expected to maximize the "enjoyment" of the vacation.

I. Introduction

Planning is a problem of defining a sequence of actions that gets one from some initial state to some goal state. Automated planning is a branch of artificial intelligence that tries to solve this problem using a computer [2]. However, there is a large class of planning problems that we meet in our daily life that is difficult for a computer to solve, not only because of the involved computational complexity, but because the goal state (as well as the consequence of individual actions) is hard or even impossible to formalize. In contrast, in many of these problems, the goal (and the effect of actions) is intuitively understandable by humans, making the planning humanly possible.

As a simple example, consider a vacation trip planning. A person may have some tentative start and end dates for her vacation, a preference of what she likes to do and a geographic area where she wants to travel. Based on this data she now needs to compile a potential set of places and attractions to visit and, from this set build a vacation schedule (essentially an ordered subset of the original set). A typical goal here may be to enjoy the vacation the most and/or to expand horizons. Such a goal is naturally subjective and hard to formalize (relevant factors may include total travel distances, attractions along the way, price and many more). However, people sharing similar taste/interests are likely to have the same notion of objective function and their experience and opinion can assist in the planning.

In general we are targeting here problems where one has a large set of items from which she needs to choose a subset and then order this subset in a sequence that will give the best value. The "value" definition is domain-specific, hard to formalize but easy to comprehend by humans. The vacation planning example above is one such instance. Another example is academic schedule planning, where the goal for instance is to obtain solid knowledge of a given subject area.

Solving such problems require *expertise* in the domain of the problem, which is often gained by experience, solving instances of the same (or similar) problems. Since many people deal with similar planning problems, it is reasonable to assume that the *crowd* may provide useful insight here. Indeed, several attempts were made in this direction. For example, for academic schedule planning, the *CourseRank* system [3] allows students to rate courses and provides a convenient tool to compile recommended courses into schedule. Another example is the *Cross-Service Travel Engine for Trip Planning* [4] that allows harvesting POIs (points of interest) from various traveling recommendation sites and provides a tool to compile a trip schedule from these POIs. Theses systems however focus on identifying the *set* of relevant items (courses, POIs), but the non-trivial task of *ordering* them in an ideal way, to form an actual plan, is left to the user.

Assisting the user in this fairly challenging task is the goal of the present work. We refer below to an ordered list of items as a *plan* and present CrowdPlanr , a system that employs the crowd to build "good" plans (w.r.t some abstract quality criteria) for specific tasks. It takes as input a set of relevant items (that can be obtained from the existing systems mentioned above) and intelligently asks users from the crowd series of simple questions (about possible 1-step continuations of given partial plans), using the answers to identify the plans preferred by the crowd.

Intuitively, the set of all possible plans (ordered lists) that can be built from a given set of items can be modeled as a tree, where each node is an item, its ancestors are the items preceding it in the plan and its children are the items that may follow it. A root-to-leaf path in this tree represents a plan. One may rate (and correspondingly rank) plans by the probability of a person to consider a given plan as the best (w.r.t to the given abstract criteria). As the size of this tree may be extremely large (exponential in the size of the items set), it is clearly impractical to ask the crowd about each possible

plan. Instead, we employ in CrowdPlanr a novel efficient algorithm that traverses this tree incrementally. It carefully restricts attention to the more promising plans - ones with highest maximum potential score (to be formally defined in the sequel) and optimally chooses at each step the 'best' questions (about possibly continuation), so that the overall number of questions that the crowd needs to be asked is minimized.

Outline of the demonstration: We demonstrate the planning problem described above, along with our solution, in the context of a question-answer game played by the ICDE'13 attendees, whose goal is to assist users in vacation planning. The vacation may be planned at different levels of granularity - countries to visit, cities within a country, attractions in a given city. We start by building a coarse-grained plan then refine it, focusing on Brisbane as an example. To motivate users to answer questions, players will be allowed to view the constructed sightseeing plans only after answering a minimal number of questions. We show how CrowdPlanr incrementally chooses what questions to ask, and how the gathered answers are analyzed so that the sightseeing plan preferred by crowd (conference attendees) is identified[1].

II. Technical Background

We begin with an informal presentation of the model underlying CrowdPlanr , then describe the algorithm used for choosing the questions to be posed to the crowd.

Complimentary components: We assume that we are given an initial finite set $\mathbb{S}$ of potential items to build a plan from. This set already reflects the preferences the user has defined when she requested a plan. We will use this set to suggest to the user possible answers when we ask a question. Some of these items may become irrelevant as we progress, which will be reflected by the users not selecting them as answers. There are multiple tools that can be used for identifying this initial set $\mathbb{S}$ of items, e.g. *TripAdvisor* [5] for vacation planning, *CourseRank* [3] for academic schedule planning. We use the former in our demo.

As a simple running example we will use below the planning of a vacation in Italy (at the city granularity), starting from Rome. The set of items $\mathbb{S}$ in this case includes commonly visited Italian cities, e.g., {*Milan, Venice, Verona, Florence, Pisa, Trento, Bologna, Naples, ...*}.

Note that, in general, not every user can answer every question. Indeed users that have never visited/read/heard vacation stories about Italy cannot help much in planning a vacation there. The targeting of questions to relevant users is by itself a challenging problem that may be addressed by a variety of methods (e.g. using semantic knowledge about users [6], employing collaborative-filtering based techniques [6], [7], etc.). In principle, any such black-box algorithm can be plugged into our system. Specifically, in our demo we will simply assume all conference participants to be relevant crowd.

[1]We note that the demo focuses only on sightseeing. Other complementary issues of trip planning (e.g. hotels selection, transportation, etc.) are naturally complementary and may be added to the plan using existing services.

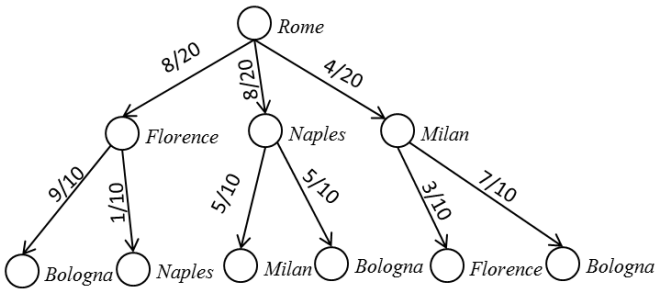

Fig. 1. An example of a tree representing a set of plans

Model: CrowdPlanr allows users to plan vacations at different levels of abstraction, zooming in and out between levels. For space constraints we focus below on a single level and explain things intuitively in this simplified context. Given a set $\mathbb{S}$ of items, a *plan* is an ordered subset of $\mathbb{S}$ [2]. We will use two special items not in $\mathbb{S}$ - $\dagger$ to mark the beginning of the plan and $\ddagger$ to mark an end of the plan. A *complete* plan is an ordered sequence of items $(\dagger, a_1, \ldots, a_k, \ddagger)$, with no repetitions, starting with a beginning marker and ending with an end marker. We also consider partial plans - prefixes that can be expanded by adding new items; these do not have an end marker. The set of all possible plans may be represented by a tree, called *a decision tree*, where the root is labeled by the start marker, internal nodes are labeled by items from $\mathbb{S}$, leaves are labeled by the end marker, and each internal node v_i represents a partial plan $p_i = (\dagger, a_1, \ldots, a_i)$, corresponding to the labels of nodes on the path from the root to v_i.

The decision tree is built incrementally by asking the crowd questions on the nodes already in the tree (at the beginning the tree consists only of the root). Given a node v_i (and its corresponding partial plan p_i) we ask the users questions of the form *"Given a plan p_i which item should we add to it?"*. The user will answer with an item to add, or indicate that the plan should stop at this item. Answers to these questions define a probability distribution on the children of every node. We use these distributions to define a score for every node - a *score of a node* is its probability to follow its parent in node's partial plan. A *score of a (partial) plan* is then a multiplication of scores of the nodes composing it.

To continue with our example, a portion of the tree describing (partial) Italy vacation plans is depicted in Figure 1. Assume that 20 questions were asked for the root and 10 for each of the other shown nodes and that the labels on incoming edges denote the nodes score. Thus the scores of following partial plans are:

- *(Rome, Florence, Bologna)* - 0.36
- *(Rome, Naples, Milan)* - 0.2
- *(Rome, Milan, Bologna)* - 0.14

The previous definitions do not place an upper bound on the number of users that need to be asked in order to compute the probability distribution for a given node. In principle we could ask all available users for each node, but this exhaustive approach can be prohibitively expensive in practice. Instead,

[2]More generally, one may also want to consider partially ordered subsets. For simplicity we ignore this here

we expect applications to place a limit on the number of obtained answers. For this purpose, we define a threshold $\mathcal{N}$ that denotes the desired number of users to be probed for a node. (This may be determined, e.g., based on the desired sampling error bounds [8].) Thus, in principle, by asking $\mathcal{N}$ questions on all of the (incrementally added) nodes (until no more new nodes are added) we can obtain a final tree $\mathcal{T}_\mathcal{O}$ from which we can find the best plan - $p_\mathcal{O}$ [3]. Note however that, since the size of this tree may be exponential in the size of the items set, it is clearly impractical to build it fully and ask the crowd about each of its nodes. Instead, we employ an efficient algorithm that intelligently traverses the tree and processes only the minimal necessary parts.

The core algorithm: The key observation underlying our solution is that finding $p_\mathcal{O}$ *exactly* is not really necessary - as the quality criteria is anyway abstract, two plans having almost the same score are reasonably equally good. Thus we define a *correct answer* to be a plan p s.t. $score(p_\mathcal{O}) - score(p) \leq \epsilon$, where ϵ is a given allowed error constant.

As we ask the crowd questions, we incrementally discover $\mathcal{T}_\mathcal{O}$ and its nodes score. Note that we can return a plan p as an answer as soon as we are sure that its score in $\mathcal{T}_\mathcal{O}$ (that has been only partially discovered so far) is at least $score(p_\mathcal{O}) - \epsilon$. To be certain that a plan p is indeed a correct answer one has to prove that there is no other plan p' in $\mathcal{T}_\mathcal{O}$ with a score significantly higher than $score(p)$. Since a full tree has not been discovered, this requires showing that in every *possible completion* of the current tree T (and the user answers obtained thus far), p has the highest score up to ϵ. The notion of *possible completion* is defined in the intuitive manner, looking at all possible trees that may be constructed by continuing asking all possible questions (and obtaining all possible combinations of answers) from the given state.

Our algorithm works in a greedy manner as follows. At each point it (1) computes for the existing partial plans their highest potential score, namely the maximal score that it can have in some possible completions of the given tree, then (2) selects a plan having maximal such potential sore, and (3) ask questions on the closest-to-the-root not-yet-exhausted node of this plan (i.e. a node where we haven't yet asked $\mathcal{N}$ questions). It stops when it finds a plan whose *minimal potential* score is smaller by less than ϵ than the maximal potential score of all other plans.

We conclude with two remarks, one regarding the complexity of the algorithm and the other regarding its optimality.

Remark 1 First, it is important to note that the maximal and minimal potential scores of a plan (and thus also the stopping condition), can be computed in time polynomial in the size of the current tree, *without having to materialize its missing parts, or consider all possible combinations of missing user answers*. Indeed one can show that a maximum potential score for a plan is achieved if from now on all the answers are in favor of that plan. Similarly, a minimum potential score is achieved if from now on all the answers are against that plan. And thus both can be calculated in $PTIME$.

[3]Recall that $\mathbb{S}$ is finite and that each item appear in a plan only once, hence $\mathcal{T}_\mathcal{O}$ is finite.

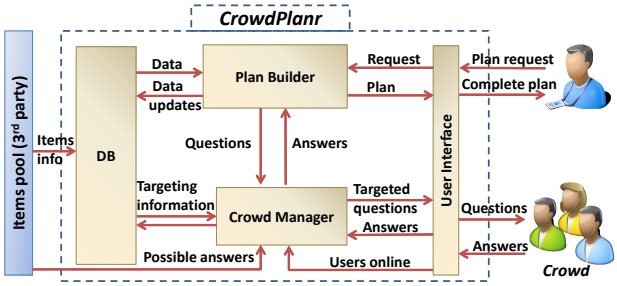

Fig. 2. CrowdPlanr architecture

To illustrate, consider again the tree depicted in Figure 1. Suppose that $\mathcal{N} = 20$, thus no node in the tree is exhausted yet (except for root). Maximum score for a plan is achieved if for all remaining questions all answers match this plan, maximum potential scores for some of the partial plans in this tree are:

- *(Rome, Florence, Bologna)* - $\frac{8}{20} \cdot \frac{19}{20} = 0.38$
- *(Rome, Naples, Milan)* - $\frac{8}{20} \cdot \frac{15}{20} = 0.3$
- *(Rome, Milan, Bologna)* - $\frac{4}{20} \cdot \frac{17}{20} = 0.17$

The plan that has the highest maximum potential score is *(Rome, Florence, Bologna)* and thus our algorithm will ask a question about highest non-exhausted node of this plan - *Florence*.

Remark 2 A dominant cost factor in crowd sourcing applications is the number of questions being asked. This is because answering may be slow and possibly incur monetary cost. We show in the full version of the paper that our algorithm is extremely efficient with that respect. Following [9], define an algorithm to be instance-optimal if for any possible input instance, its cost of operation (in terms of number of questions posed to the crowd) is at most the same order of magnitude as that of any other correct algorithm. Indeed, we are able to show that our algorithm is instance-optimal. We omit the proof for space constraints.

Related Work: Using crowd as a source of knowledge has become popular in the last years. Some research (e.g. [10]) deals with the development of a unified language and model to allow data collection from both humans and machines while other (e.g. [11], [12]) concentrate on ways to process this data, clean it and extract insights from it. Other related work (e.g. [7], [13]) is directed to optimizing expected cost (number of questions) and expected error. Our work is also concerned with minimizing the number of questions, but requires a different approach as the result consists of an ordered list, entailing inherent dependency between questions and answers.

III. System Overview

CrowdPlanr is implemented in PHP and uses a MySQL database. Figure 2 illustrates the system architecture.

Plan Builder receives requests for building a plan and executes our algorithm to build it. Questions generated by the algorithm are forwarded to the *Crowd Manager*. Once the algorithm decides to stop, plan is returned to the requesting user.

Crowd Manager constantly receives an updated list of online users from *UI* and targeting information from the *DB*. Once a question is received from the *Plan Builder*, it is enriched with

Fig. 3. CrowdPlanr user interface

a set of possible answers from the *Items pool* and presented to a selected user via *UI*. Received answer is passed back to *Plan Builder*.

DB stores answers collected from the crowd, targeting information and general information about the locations.

Items pool is completely 3rd party and stores information about the locations and used for initial filtering of the options and possible answers.

UI module is used to interact with users (both requesting a plan and contributing to it). Figure 3 shows a screenshot of this operation. In this example user from crowd faced with question "Trip in Australia", where two previous places that already proposed were (Canberra, Australia) and (Sydney, Australia).

CrowdPlanr can also run as a service, by disabling UI and invoking it with system calls, which is useful for web sites that want to keep their own UI.

IV. Demonstration

In this demonstration, CrowdPlanr will engage the ICDE13 attendees to build a comprehensive crowdsourced database of travel plans at different granularity levels (countries, cities, attractions in a given city) . This will be done via a multiple-choice questions game where users are shown partial travel plans, along with a set of possible continuation, and are asked to select the best one (according to given abstract criteria). The list of possible answers is predefined (implemented via user-friendly auto-completion mechanism) but not limited, and users can also type in new place to visit. The generated database consists, correspondingly, of travel plans for different locations in the world, including in particular Australia and Brisbane. To encourage participation, players will be allowed to view the constructed travel plans and request new ones, but only after they themselves answer a minimal set of questions per such view.

The data set used in the demo will be initialized with a set of geographical locations/attractions divided to three categories: countries, cities in those countries and attraction places in those cities. In order to not start the demo with an empty set of travel plans we will partially fill it by fake, yet realistic (partial) plans, using information extracted from TripAdvisor. We will also add to the database a list of ICDE'13 participants, extracted from the conference program, including the cities and countries of the participants universities. This will allow CrowdPlanr to target questions on given countries to corresponding users (if such currently play).

We start the demonstration by explaining the game, its goal and rules. Then, we let our audience play it on several laptops allocated for that. In parallel, we will explain how CrowdPlanr works (repeatedly, so that players who finished

Fig. 4. Statistics page

playing can join, allowing others to play). We will show, on an Administrator screen, the current state of the data: how many (partial) vacation plans exist, how many user answers we already collected, which questions were recently asked, and what are the current (partial) plans with potential high score. An example for one of the Administrator views that we will show is depicted in Figure 4. We then request one of the attendees that played the game earlier to log-in again into the system (users log-in with their real name). We will first view her previous answers (if she agrees) then start the game again and follow its course. We will examine the questions that CrowdPlanr poses to the user and her answers, and reveal, in parallel, on an Administrator screen, what is happening under the hood. We will describe our algorithm and explain why these specific questions were chosen by the system. We will also explain the effect that the given answers have on the system's state and how the system determines that sufficient information has been collected to determine the best plan for a given request.

Acknowledgment

This work has been partially funded by the European Research Council under the European Community's Seventh Framework Programme (FP7/2007-2013) / ERC grant MoDaS, agreement 291071, by the Israel Ministry of Science, and by the US-Israel Bi national Science foundation.

References

[1] A. J. Quinn and B. B. Bederson, "Human computation: a survey and taxonomy of a growing field," in *CHI*, 2011, pp. 1403–1412.
[2] M. Ghallab, D. S. Nau, and P. Traverso, *Automated planning - theory and practice*. Elsevier, 2004.
[3] "Courserank," http://courserank.stanford.edu/.
[4] G. Chen, C. Liu, M. Lu, B. C. Ooi, S. Ying, A. Tung, D. Zhang, and M. Zhang, "A cross-service travel engine for trip planning," in *SIGMOD*, 2011, pp. 1251–1254.
[5] "Tripadvisor," http://www.tripadvisor.com/.
[6] G. Adomavicius and A. Tuzhilin, "Towards the next generation of recommender systems," *IEEE TKDE*, 2005.
[7] R. Boim, O. Greenshpan, T. Milo, S. Novgorodov, N. Polyzotis, and W.-C. Tan, "Asking the right questions in crowd data sourcing," in *ICDE*, 2012, pp. 1261–1264.
[8] R. Groves, F. J. Fowler, M. Couper, J. Lepkowski, E. Singer, and R. Tourangeau, *Survey Methodology*. John Wiley and Sons, 2009.
[9] R. Fagin, A. Lotem, and M. Naor, "Optimal aggregation algorithms for middleware," *J. Comput. Syst. Sci.*, vol. 66, no. 4, pp. 614–656, 2003.
[10] A. G. Parameswaran and N. Polyzotis, "Answering queries using humans, algorithms and databases," in *CIDR*, 2011, pp. 160–166.
[11] D. Deutch, O. Greenshpan, B. Kostenko, and T. Milo, "Using markov chain monte carlo to play trivia," in *ICDE*, 2011, pp. 1308–1311.
[12] M. J. Franklin, D. Kossmann, T. Kraska, S. Ramesh, and R. Xin, "Crowddb: answering queries with crowdsourcing," in *SIGMOD*, 2011.
[13] A. G. Parameswaran, H. Garcia-Molina, H. Park, N. Polyzotis, A. Ramesh, and J. Widom, "Crowdscreen: algorithms for filtering data with humans," in *SIGMOD*, 2012, pp. 361–372.