Finding Dependencies Between Actions Using the Crowd

Walter S. Lasecki[1], **Leon Weingard**[1], **George Ferguson**[1], **and Jeffrey P. Bigham**[2,1]

Computer Science, ROC HCI[1]
University of Rochester
{wlasecki,weingard,ferguson}@cs.rochester.edu

Human-Computer Interaction Institute[2]
Carnegie Mellon University
jbigham@cmu.edu

ABSTRACT

Activity recognition can provide computers with the context underlying user inputs, enabling more relevant responses and more fluid interaction. However, training these systems is difficult because it requires observing every possible sequence of actions that comprise a given activity. Prior work has enabled the crowd to provide labels in real-time to train automated systems on-the-fly, but numerous examples are still needed before the system can recognize an activity on its own. To reduce the need to collect this data by observing users, we introduce ARchitect, a system that uses the crowd to capture the dependency structure of the actions that make up activities. Our tests show that over seven times as many examples can be collected using our approach versus relying on direct observation alone, demonstrating that by leveraging the understanding of the crowd, it is possible to more easily train automated systems.

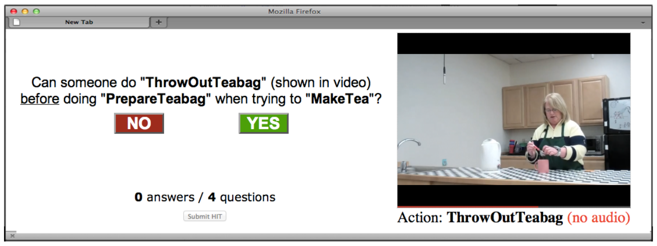
Author Keywords
Crowdsourcing; activity recognition; constraint finding

ACM Classification Keywords
H.4.2 Info. Interfaces & Presentation: User Interfaces

INTRODUCTION

Activity recognition (AR) lets interactive systems work better by allowing them to consider what users are doing when deciding what assistance to provide. Training AR is costly because activities can be done in a variety of ways, and robustly training an automated system requires it to have seen all of the different ways an activity can be performed. Prior work has allowed the crowd to label activities to train such systems on-the-fly when an unknown sequence of actions is encountered [4], however many examples are still needed before the system can recognize an activity without assistance.

In this paper, we present ARchitect, a system that enables the crowd to capture the structure of observed activities in the format of a dependency graph. We focus on activity recognition in the home to provide timely, task-relevant information and support to those who need it.



Can someone do "**ThrowOutTeabag**" (shown in video) before doing "**PrepareTeabag**" when trying to "**MakeTea**"?

0 answers / **4** questions

Action: **ThrowOutTeabag** (no audio)

Figure 1. ARchitect's interface for finding relationships between actions. Workers are asked if the action shown requires a previous action to be performed first, then responses are aggregated to generate a dependency graph.

For example, prompting systems can keep people with cognitive disabilities on track [9], and smart homes can detect when to summon help so that older adults can safely live independently longer [8].

ARchitect elicits a dependency graph of actions forming high-level activities from the crowd, and gleans more information from a single video than automated observation alone. ARchitect's interface (Figure 1) asks workers to mark which actions must be performed before others for the action to make sense (their dependencies). For instance, it doesn't make sense to heat a kettle before putting water in it. From this input, ARchitect generates a dependency graph that defines all valid orderings of the actions composing an activity. Our experiments with 288 Mechanical Turk workers demonstrate that ARchitect can identify 22 valid ways to complete an activity from only 3 observations of the activity being performed. This lays the groundwork for systems that use the crowd's understanding of problems and their context to more efficiently and robustly train automated systems by formalizing knowledge.

BACKGROUND

Prior work has explored augmenting automated systems by creating hybrid systems that can decide when to route a task to a human or machine [6], and by learning from multiple contributors [7]. Legion:AR [4] uses the crowd to support and train a deployed activity recognition system in real-time by having workers generate labels for a video stream. These labels are then used to train a Hidden Markov Model (HMM) based activity recognition system that uses active learning to decide when to query the crowd. It can also suggest labels to the crowd as it learns to better recognize activities. This enables systems to scale from crowd-powered to fully automated.

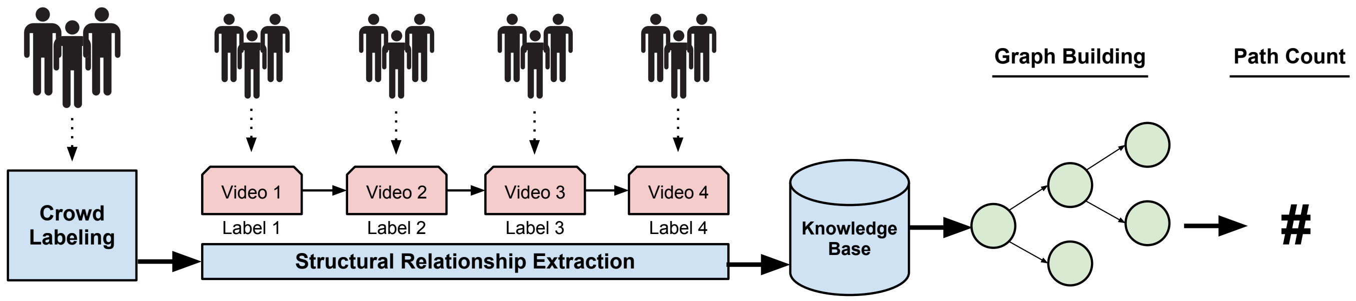


Figure 2. ARchitect system diagram. First, video is decomposed by the crowd into a set of videos, each containing a single mid-level action, then an action label is found. Next, ARchitect recruits crowd workers for each labeled video segment (action). These workers are asked a series of randomly ordered questions pertaining to the required ordering of pairs of tasks. Finally, the set of pairwise restrictions are used to create a dependency graph. This information can be used to help the system recognize familiar actions performed in orders it has never seen before.

This paper extends the idea of learning basic activity labels from the crowd to learning relational aspects of activities including dependencies between actions. Cascade [2] is a system that uses the crowd to extract structured taxonomies from unstructured datasets, but, unlike ARchitect, is not able to infer temporal structure.

SYSTEM

ARchitect has three main components: labeling, dependency finding, and alternate order generation (Figure 2). To find the segment times and labels, ARchitect can use the crowd via Legion:AR [4], or any other existing labeling method. Our experiments use a 'Tea Making' domain containing the following 6 action labels:

- `FillKettle` Tea kettle is filled with water from faucet.
- `GetIngredients` Teabag and cup gotten from cabinet.
- `PrepareTeabag` Wrapper is opened; teabag removed.
- `ReturnIngredients` Teabag box returned to cabinet.
- `PourHotWater` Water boiled in kettle; poured into cup.
- `SteepTea` Teabag is allowed to soak in boiling water.
- `ThrowOutTeabag` Teabag moved from cup to trash.

Finding Preceding Actions

To determine the dependencies between actions, we ask workers to answer a set of questions for a series of video segments (shown in Figure 1). These questions are generated from the labeled actions the system has observed up to that point. The segments are gleaned from a larger video of the complete task and limited in scope to keep the task for each segment manageable for workers.

Asking the Right Questions

ARchitect asks workers a series of 'Yes or No' questions. We initially tried asking questions of the form 'Is it important to do [**action_in_question**] before [**action_in_video**] to complete [**high_level_activity**]?'. However, asking the question in this way led users to answer too liberally, saying that everything was important to do in the expected order. After a number of iterations, we found that reversing the question and asking workers 'Is it possible to do [**action_in_video**] before [**action_in_question**] when doing [**high_level_activity**]?', resulted in a balance between the scope of the question and the willingness of workers to introduce constraints.

Providing Context

The question posed to workers includes the name of the activity that the action belongs to, to avoid cases where the individual actions do not provide enough context to determine if a constraint exists. We also present the workers with a video of the activity because we want results that are specific to the instance of an action being observed, not just speculation about the generalized activity. For instance, the `FillKettle` action can involve getting water from the tap or from the items collected in `GatherIngredients`. Without seeing the video, workers would not have known which case applied and may have been split on which option to choose.

Generating the Dependency Graph

From the crowd's input, we can create a dependency graph – a directed acyclic graph (DAG) – to represent the relational structure of the actions. For each 'Yes' answer given, we add 1 to the weight of the directed edge going from the prerequisite action to the given action (or add a new edge if needed). The resulting dependency graph explicitly captures all of the pairwise constraints between actions. We can then simplify this graph by removing all edges that form a second path from a requisite action to another action, keeping only the longest path between any two nodes u and v in the graph.

To filter out bad input, we require agreement between more than half of the workers participating in an average task (for that trial) to include an edge. The filtering threshold can be set to match the specifics of the crowd being used (size and expected worker reliability). We apply this filtering step *before* simplifying the graph to avoid losing edges between connected nodes in the final version. We could normalize the edge weights and interpret them as probabilities, but we do not here because our goal is to recreate the training data that would be generated by an expert.

To calculate how many valid possible execution paths exist, we generate a list of all possible (complete) action orderings that do not contain a dependent action as a parent of their prerequisite. Path counting provides a measure of the potential learning difference between direct observation and ARchitect.

EXPERIMENTS

To validate ARchitect, we are interested in testing two aspects: (i) the crowd's ability to accurately identify dependencies in observed action orderings, and (ii) the validity of the resulting activity execution paths found using this approach. The latter allows us to better understand the tradeoff in accuracy versus the cost savings of requiring fewer observations of a given activity to reliably identify it later. We used expert-generated labels and directly measured the dependency graphs output by the system to avoid confounding our results with variation in the crowd labeling process, or the selection and implementation details of an HMM-based AR system.

To create our tea-making dataset, we recorded a video of 3 participants stepping through the process of making tea. The kitchen was configured the same at the beginning of the experiment for each participant, but participants were not asked to take a particular course of action, so long as they successfully demonstrated how to make a cup of tea. We chose to use 3 examples from the same domain to better explore how crowd responses may be merged (discussed later).

We then collected between 1 and 5 dependency labels from each of 288 unique Mechanical Turk workers. Their results were compared to our gold standard generated by 3 researchers. The inter-rater reliability was very high as indicated by a Fleiss' Kappa score of $k = .92$. In order to help workers understand the task they were being asked to perform, we first asked each worker to mark dependencies in two examples, and then provided feedback as to why the provided labels were right or wrong.

Identifying Action Dependencies

We first test how accurately ARchitect can extract individual relationships from a video of a given activity (Figure 3). In order to use only action dependencies with high confidence, we filter the data so that a link between two nodes is only included if multiple workers agreed on it. To more easily visualize the constraints, we then remove redundant paths, keeping only the longest path between any two actions. Note that while this filter operation is performed on an un-reduced graph containing the original set of edges, the reduced version (shown in Figures 3 and 4) is equivalent in terms of the constraints expressed, so there is no effect on the resulting path count.

Figure 5 shows the change in precision and recall as the filter threshold is increased, making the filtering rule stricter. These curves meet exactly at 100% in Videos #2 and #3, meaning that no valid edges were removed as the filter strength was increased until all invalid edges were removed. In Video #1 we can see a case where this is not true, but the small size of this gap indicates a minor error.

Merging Results from Multiple Users' Sessions

To generalize the results we collect with ARchitect, we combine results from multiple different user sessions into a single graph for the underlying activity. To do this, we

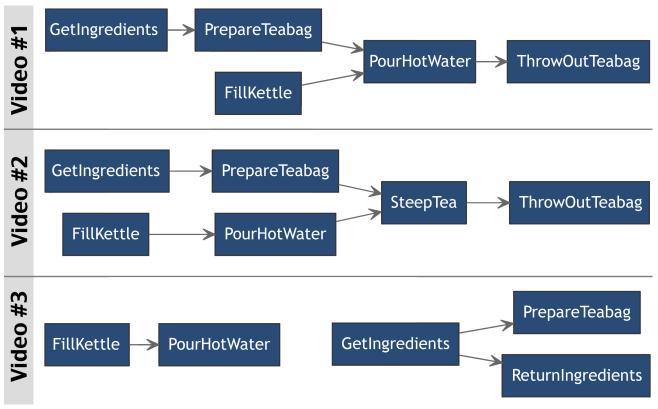


Figure 3. The dependency graphs output by our 3 trials. An average of 22.3 workers were asked each question. Thresholds of 50%, 50%, and 40% were used, respectively.

take each of the filtered graph results and normalize the edge weights. We then merge the two graphs by aligning the matching actions in each, and connecting or adding the remaining actions, as needed. If an edge agrees with another, the weights are summed in the new graph. This gives us a weighted set of all pairwise constraints identified by the crowd.

Next, we run another filtering step in which we remove the edges with less than 0.5 weight (those that don't have majority agreement). This prevents the combined dependency graph from being over-constrained. The final output of our tea making example is shown in Figure 4. In the end, 22 unique valid task executions were identified from observing just 3 actual instances. This means we have collected over 7 times as much training data from our videos by using the crowd.

Finding Execution Paths

Next, we compute the number of orderings of all of the observed actions that do not violate the constraints in the graph. Increasing the agreement required in the filtering step results in fewer included edges. The constraints on the activity are thus reduced, potentially increasing the number of valid paths. Conversely, decreasing required agreement will allow more constraints to be added, reducing the number of valid paths (in the limit, only the observed ordering is valid). The maximum number of possible valid paths increases exponentially as fewer constraints are included. For Video #1 there were at most 120 distinct paths, for Video #2 there were 720, and for Video #3 there were again 120. We found that an average of 10.7 possible action orderings could be identified for each of our videos, compared to only one with conventional approaches.

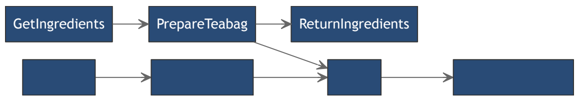


Figure 4. Final dependency graph generated by merging the results of observing 3 users.

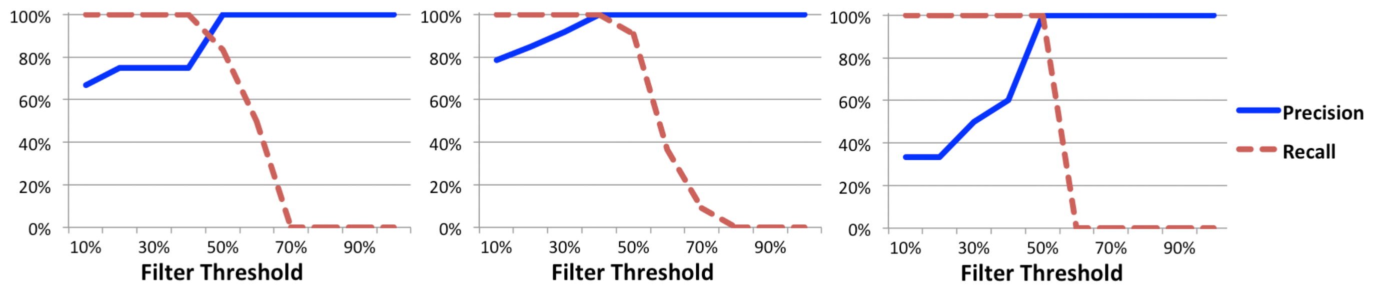


Figure 5. Precision and recall of edges plotted over increasing thresholds for our videos. Precision increases with required agreement at first as invalid constraint edges are removed. Recall eventually decreases as even valid edges are removed.

DISCUSSION

Our results show that ARchitect is able to accurately extract structural data using the crowd. Despite individual workers introducing errors, the aggregate result is a set of constraints ranked reliably by confidence.

Removing Ambiguity

One determining factor in the level of consistency is the ambiguity in the question and action labels. We present workers with a video of the current action to show exactly how the user performed it, but we describe prior actions using only text to avoid overloading workers with multiple videos. One way to correct for this is to show workers what lower-level actions make up the mid-level actions they are shown. For example, in the `GatherIngredients` step, it is unclear if water is considered an ingredient (in our experiments it was not – the kettle is filled directly form the faucet in the `FillKettle` step). But if we include low-level actions such as 'turn on faucet', then it is evident that the sub-task has been completed during the action.

It is also possible to re-recruit some workers who have answered questions about previous actions in the same activity, to leverage their knowledge of past actions. Despite often performing dozens or hundreds of tasks per hour, workers do retain task-specific knowledge from the past [3]. By encouraging workers to join future tasks (e.g., by offering a bonus), we reduce the ambiguity in labeling prior tasks, while not requiring workers to view multiple videos in a single step, keeping latency low.

Limitations

Our study focused on a single well-known domain in activity recognition, meaning we don't explore the whole range of activity and action types that would be encountered in more open-ended domains. We also extract a straightforward dependency structure instead using of a more expressive representation, such as a temporal logic [1]. However, our work aims to show that this type of information can be useful to automated systems if properly elicited from the crowd.

CONCLUSIONS AND FUTURE WORK

ARchitect presents the first example of what is possible by using on-demand human intelligence in the knowledge acquisition processes. A similar approach might be used to extract more activity relationship information. For example, we could use the crowd to identify the latent states that make prior actions necessary. This could allow us to define planning operators, and thus construct more robust representations of action dependence, such as a planning graph. We might also begin to integrate contributions from the AR system itself by adding its predictions of valid action orders to the graph generated by the crowd. Using ARchitect, we have shown that we can extract structured knowledge quickly, suggesting a future in which automated systems can be trained on-the-fly from one-off examples.

ACKNOWLEDGMENTS

This work was supported by National Science Foundation awards #IIS-1149709 and #IIS-1218209, and a Microsoft Research Ph.D. Fellowship.

REFERENCES

1. Allen, J. F. Towards a general theory of action and time. *Artificial intelligence* 23 (2). 123–154. 1984.
2. Chilton, L. B., Little, G., Edge, D., Weld, D. S., and Landay, J. A. Cascade: Crowdsourcing taxonomy creation. *CHI 2013*.
3. Lasecki, W.S., White, S., Murray, K.I., and Bigham, J. P. Crowd memory: Learning the collective. *CI 2012*.
4. Lasecki, W. S., Song, Y. C., Kautz, H., and Bigham, J. P. Training activity recognition systems online using real-time crowdsourcing. *CSCW 2012*.
5. Little, G., Chilton, L. B., Goldman, M., and Miller, R. C. Turkit: human computation algorithms on mechanical turk. *UIST 2010*.
6. Quinn, E. J., Bederson, B. B., Yeh, T., and Lin, J. Crowdflow: Integrating machine learning with mechanical turk for speed-cost-quality flexibility. Tech. Rep., University of Maryland, 2010.
7. Richardson, M., and Domingos, P. Learning with knowledge from multiple experts *ICML 2003*.
8. Tapia, E., Intille, S., and Larson, K. Activity recognition the home using simple and ubiquitous sensors. *Pervasive 2004*.
9. Zhao, L., Sukthankar, G., and Sukthankar, R. Robust active learning using crowdsourced annotations for activity recognition. *HC 2011*.