TextToHBM: A Generalised Approach to Learning Models of Human Behaviour for Activity Recognition from Textual Instructions

Kristina Yordanova
University of Rostock
18059 Rostock
Germany

Abstract

There are various knowledge-based activity recognition approaches that rely on manual definition of rules to describe user behaviour. These rules are later used to generate computational models of human behaviour that are able to reason about the user behaviour based on sensor observations. One problem with these approaches is that the manual rule definition is time consuming and error prone process. To address this problem, in this paper we outline an approach that learns the model structure from textual sources and later optimises it based on observations. The approach includes extracting the model elements and generating rules from textual instructions. It then learns the optimal model structure based on observations in the form of manually created plans and sensor data. The learned model can then be used to recognise the behaviour of users during their daily activities. We illustrate the approach with an example from the cooking domain.

Introduction

Some activity recognition (AR) approaches utilise human behaviour models (HBM) in the form of rules. These rules are used to generate probabilistic models with which the system can infer the user actions and goals (Hiatt, Harrison, and Trafton 2011; Ramirez and Geffner 2011; Krüger et al. 2014). Such types of models are also known as computational state space models (CSSM) (Krüger et al. 2014). They treat activity recognition as a plan recognition problem, where given an initial state, a set of possible actions, and a set of observations, the executed actions and the user goals have to be recognised (Ramirez and Geffner 2011). These approaches rely on prior knowledge to obtain the context information needed for building the user actions and the problem domain. The prior knowledge is provided by a domain expert or by the model designer. This knowledge is then used to manually build a CSSM. The manual modelling is however time consuming and error prone (Nguyen, Kambhampati, and Do 2013).

To address this problem, different works propose the learning of models from sensor data (Zhuo and Kambhampati 2013). One problem these approaches face is that sensor data is expensive (Ye, Stevenson, and Dobson 2014). Furthermore, sensors are sometimes unable to capture fine-grained activities (Chen et al. 2012), thus, they might potentially not be learned.

To reduce the need of domain experts and / or sensor data, one can substitute them with textual data (Philipose et al. 2004). More precisely, one can utilise the knowledge encoded in textual instructions to learn the model structure. Textual instructions specify tasks for achieving a given goal without explicitly stating all the required steps (Branavan, Zettlemoyer, and Barzilay 2010). On the one hand, this makes them a challenging source for learning a model (Branavan, Zettlemoyer, and Barzilay 2010). On the other hand, they are usually written in imperative form, have a simple sentence structure, and are highly organised. Compared to rich texts, this makes them a better source for identifying the sequence of actions needed for reaching the goal (Zhang et al. 2012).

According to (Branavan et al. 2012), to learn a model of human behaviour from textual instructions, the system has to: 1. **extract the actions' semantics** from the text, 2. **learn the model semantics** through language grounding, 3. and, finally, to **translate it into computational model of human behaviour** for planning problems. To address the problem of learning models of human behaviour for AR, we extend the steps proposed by (Branavan et al. 2012). We add the need of 4. **learning the domain ontology** that is used to abstract and / or specialise the model. We also replace step 3. (models for planning problems) with **computational models for activity recognition** as the targeted model format, as they are able to reason about the human behaviour based on noisy or ambiguous observations (Hiatt, Harrison, and Trafton 2011; Ramirez and Geffner 2011).

The contribution of this paper is twofold: (1) we present an approach for learning HBM from textual instructions. In difference to existing approaches for language grounding, our approach learns a complex domain ontology that is later used to generalise or specialise the model; (2) it is the first attempt at learning CSSMs for activity recognition from textual instructions. In the following we outline our approach for learning HBM for AR and illustrate it with an example from the kitchen domain. This work is based on the extended abstract in (Yordanova 2016).

Related work

There are various approaches to learning models of human behaviour from textual instructions: through grammatical patterns that are used to map the sentence to a machine understandable model of the sentence (Zhang et al. 2012; Branavan et al. 2012); through machine learning techniques (Sil and Yates 2011; Chen and Mooney 2011); or through reinforcement learning approaches that learn language by interacting with an external environment (Branavan et al. 2012; Branavan, Silver, and Barzilay 2011).

Models learned through model grounding have been used for plan generation (Li et al. 2010; Branavan et al. 2012), for learning the optimal sequence of instruction execution (Branavan, Zettlemoyer, and Barzilay 2010), for learning navigational directions (Chen and Mooney 2011), and for interpreting human instructions for robots to follow them (Kollar et al. 2014; Tenorth, Nyga, and Beetz 2010). To our knowledge, any attempts to apply language grounding to learning models for AR rely on identifying objects from textual data and do not build a computational model of human behaviour (Ye, Stevenson, and Dobson 2014). This, however, suggests that models learned from text could be used for AR tasks.

Existing approaches that learn human behaviour from text make simplifying assumptions about the problem, making them unsuitable for more general AR problems. More precisely, the preconditions and effects are learned through explicit causal relations, that are grammatically expressed in the text (Li et al. 2010; Sil and Yates 2011). They however, either rely on initial manual definition to learn these relations (Branavan et al. 2012), or on grammatical patterns and rich texts with complex sentence structure (Li et al. 2010). They do not address the problem of discovering causal relations between sentences, but assume that all causal relations are expressed within the sentence (Tenorth, Nyga, and Beetz 2010). They also do not identify implicit relations.

However, to find causal relations in instructions without a training phase, one has to rely on alternative methods, such as time series analysis (Yordanova 2015a). Moreover, the initial state is manually defined and there are only a few works that identify possible goals based on the textual instructions (Zhang et al. 2012; Babeş-Vroman et al. 2012). This limits the approaches to a predefined problem and does not allow the reasoning about different situations and goals. This is, however, an important requirement for any assistive system that relies on activity recognition.

Furthermore, they rely on manually defined ontology, or do not use one. However, one needs an ontology to deal with model generalisation problems and as a means for expressing the semantic relations between model elements.

Moreover, there have been previously no attempts at learning CSSMs from textual instructions. Existing CSSM approaches rely on manual rules' definition to build the preconditions and effects of the models. For example, (Hiatt, Harrison, and Trafton 2011) use the cognitive architecture ACT-R while other approaches rely on a PDDL[1]-like notations to describe the possible actions (Ramirez and Geffner 2011; Krüger et al. 2014). In that sense, our work is the first attempt at learning CSSMs from texts.

In this work we represent the rules in a PDDL-like notation in the form described in (Yordanova and Kirste 2015).

[1]**P**lanning **D**omain **D**efinition **L**anguage

Approach

Identifying text elements of interest

To extract the text elements that describe the user behaviour, the user actions and their relations to other entities in the environment have to be identified. This is achieved through assigning each word in a text the corresponding part of speech (POS) tag. Furthermore, the dependencies between text elements are identified through dependencies parser. To identify the human actions, the verbs from the POS-tagged text are extracted. We are interested in present tense verbs, as textual instructions are usually written in present tense, imperative form. We also extract any nouns that are direct objects to the actions. These will be the objects in the environment with which the human can interact. Furthermore, we extract any nouns that are in conjunction to the identified objects. These will have dependencies to the same actions, to which the objects with which they are in conjunction are dependent. Figure 1 gives an example of a sentence and the identified elements based on the POS-tags and dependencies.

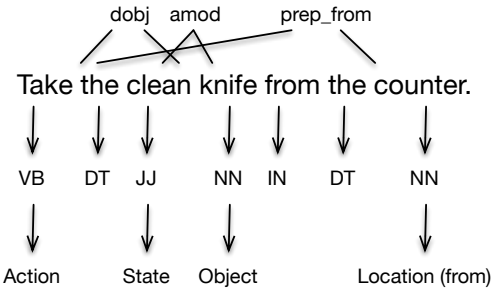

Figure 1: Elements of a sentence necessary for the model learning.

Moreover, any preposition relations such as *in*, *on*, *at*, etc. between the objects and other elements in the text are identified. These provide spacial or directional information about the action of interest. For example, in the sentence *"Put the apple on the table."* our action is *put*, while the object on which the action is executed is *apple*. The action is executed in the location *table* identified through the *on* preposition. Finally, we extract "states" from the text. The state of an object is the adjectival modifier or the nominal subject of an object. As in textual instructions the object is often omitted (e.g. *"Simmer (the sauce) until thickened."*), we also investigate the relation between an action and past tense verbs or adjectives that do not belong to an adjectival modifier or to nominal subject, but that might still describe this relation. The states give us information about the state of the environment before and after an action is executed.

Extracting causal relations from textual instructions

To identify causal relations between the actions, and between states and actions, we use an approach proposed in (Yordanova 2015a). It transforms every word of interest in the text into a time series and then applies time series analysis to identify any causal relations between the series. More precisely, each sentence is treated as a time stamp in the time series. Then, for each word of interest, the number of occurrences it appears in the sentence is counted and stored as element of the time series with the same index as the sentence index. We generate time series for all actions and for all states that change an object. To discover causal relations based on the time series, we apply the Granger causality test. It is a statistical test for determining whether one time series is useful for forecasting another. More precisely, Granger testing performs statistical significance test for one time series, "causing" the other time series with different time lags using auto-regression (Granger 1969). The causality relationship is based on two principles. The first is that the cause happens prior to the effect, while the second states that the cause has a unique information about the future values of its effect. Given two sets of time series x_t and y_t, we can test whether x_t Granger causes y_t with a maximum p time lag. To do that, we estimate the regression $y_t = a_o + a_1 y_{t-1} + ... + a_p y_{t-p} + b_1 x_{t-1} + ... + b_p x_{t-p}$. An F-test is then used to determine whether the lagged x terms are significant.

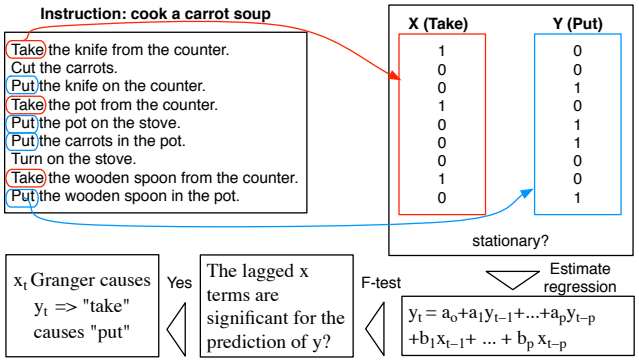

Figure 2: The procedure for discovering causal relations.

Figure 2 shows the procedure of converting text elements into time series and using them to discover causal relations.

Building the domain ontology

The domain ontology is divided into argument and action ontology. The argument ontology describes the objects, locations, and any elements in the environment that are taken as arguments in the actions. The action ontology represents the actions with their arguments and abstraction levels.

To learn the argument ontology, a semantic lexicon (e.g. WordNet (Miller 1995)) is used to build the initial ontology. As the initial ontology does not contain some types that unify arguments applied to the same action, the ontology has to be extended. To do that, the prepositions with

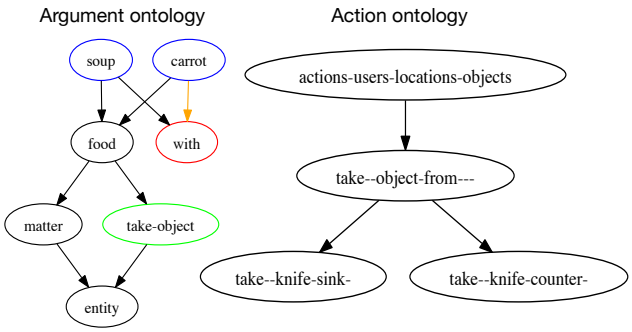

Figure 3: Argument ontology (left): objects identified through POS-tagging and dependencies (blue); hierarchy identified through WordNet (black); types identified through the relations of objects to prepositions (red); types identified based on similar preconditions (green); types identified through action abstraction (yellow). Action ontology (right): abstract representation of an action (uppermost layer); abstract representation of action *take*; concrete instances of action *take* (bottom layer).

which actions are connected to indirect objects are also extracted (e.g. *in*, *on*, etc.). They are then added to the argument ontology as parents of the arguments they connect. In that manner, the locational properties of the arguments are described (e.g. *water* has the property to be *in* something). During the learning of the action templates and their preconditions, additional parent types are added to describe objects used in actions that have the same preconditions. Furthermore, types that are not present in the initial ontology, but which objects are used only in a specific action, are combined in a common parent type. Figure 3 (left) shows an example of an argument ontology. To learn the action ontology, the process proposed in (Yordanova and Kirste 2015) is adapted for learning from textual data. Based on the argument ontology, the actions are abstracted by replacing the concrete arguments with their corresponding types from an upper abstraction level. In that manner, the uppermost level will represent the most abstract form of the action. For example, the sentence *"Put the apple on the table."* will yield the concrete action *put_apple_table*, and the abstract action *put_object_location*. Figure 3 (right) shows an example of an action ontology. This representation is used as a basis for the precondition-effect rules that describe the actions.

Generating precondition-effect rules

The next step in the process is the generation of precondition-effect rules that describe the actions and the way they change the world. The basis for the rules is the action ontology. Each abstract action from the ontology is taken and converted to an action template that has the form shown in Figure 4. Basically, the action name is the first part of the abstract entity *put_object_location*, while the two parameters are the second and the third part of the entity. Furthermore, the default predicate *(executed-action)* is added to both the precondition and the effect, whereas in the precondition it is negated.

```
(:action put
  :parameters (?o - object ?to - location)
  :precondition (and
          (not (executed-put ?o ?to)))
  :effect    (and
          (executed-put ?o ?to))
)
```

Figure 4: Example of an action template *put* in the PDDL notation.

Now the causal relations extracted from the text are used to extend the actions. The execution of each action that was identified to cause another action is added as a precondition to the second action. For example, to execute the action *put*, the action *take* has to take place. That means that the predicate *executed-take ?o* has to be added to the precondition of the action *put*. Furthermore, any states that cause the action are also added in the precondition. For example, imagine the example sentence is extended in the following manner: *"If the apple is ripe, put the apple on the table."* In that case the state *ripe* causes the action *put*. For that reason the predicate *(state-ripe)* will also be added to the precondition. This procedure is repeated for all available actions. The result is a set of candidate rules that describe a given behaviour.

As it is possible that some of the rules contradict each other, a refinement step is added. This is done by converting the argument ontology to the corresponding PDDL format to represent the type hierarchy. The initial and goal states are then generated by assigning different combinations of truth values to the set of predicates. Different combinations of initial-goal states pairs are generated from the sets of initial and goal states. Later, an initial-goal state pair as well as the rules and the type hierarchy are fed to a planner and any predicates that prevent the reaching of the goal are removed from the preconditions. This results in a set of candidate models from which the optimal model will be selected.

Learning the optimal model structure

As the model will be applied to activity recognition tasks, it is important to learn a model structure that optimises the probability of selecting the correct action[2]. To achieved that, two steps are followed (see Figure 5). First, the model is optimised based on its ability to explain existing plans describing the user behaviour. This approach is similar to the methods for model learning through observations (Branavan, Silver, and Barzilay 2011; Goldwasser and Roth 2014). Here, the observations are provided in the form of manually produced plans. The plans are obtained by asking different persons to provide a plan based on textual description of a given task. Models that are not able to predict the plan, receive no reward. From the remaining set of models, those which maximise the probability of executing the plan above a given threshold are selected. The probability is calculated based on Formula 1.

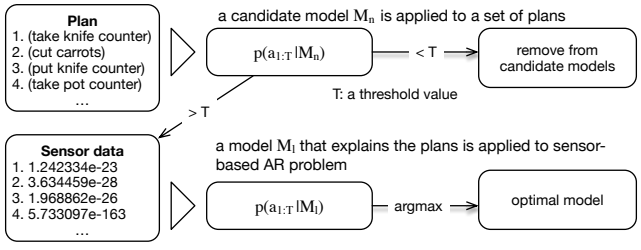

Figure 5: Learning the optimal model for a given situation based on observations.

$$p(a_{1:T} \mid M) = \prod_{t \in T} p(a_t \mid M) \tag{1}$$

$$p(a_t \mid M) = \begin{cases} 1 - p_{stop}, & \text{same action} \\ p_{stop} \times \frac{\exp(\sum_{k \in \mathcal{K}} \lambda_k f_k(a_t))}{\sum_{a \in \mathcal{A}} \exp(\sum_{k \in \mathcal{K}} \lambda_k f_k(a))}, & \text{new action} \end{cases} \tag{2}$$

M is the model used to explain the plan, a_t is the action executed at time t, k is a set of features, f is an action selection heuristic, and λ is its weight. The action selection heuristics are goal distance, landmarks, cognitive heuristics, etc (Yordanova and Kirste 2015).

After selecting the set of most promising models, they are further optimised. This is done by testing their ability to recognise activities and goals based on sensor observations. As a base for this step, the validation steps from the development process proposed in (Yordanova and Kirste 2015) are used. Formula 1 is once again used to select the model that best explains the observations.

TextToHBM: an Example

To illustrate the approach, we take as an example an experiment description of a person who is cooking a carrots soup. A description of the experiment can be found in (Krüger et al. 2014) and the sensor dataset itself in (Krüger et al. 2015). This could be considered as a simplified example, as the textual instruction contains explicit description of each execution step. Table 1 shows an excerpt of the instructions provided for executing the experiment. First, all actions in the dataset are identified[3]. For the carrots soup example, 15 actions were identified. Furthermore, all arguments are identified. For this example, 19 arguments were identified one of which was incorrectly labeled as noun (the verb "wash"). Five of the arguments serve as locations (e.g. "counter", "stove" etc.) describing places where actions are executed. The rest are objects upon which the action is executed (e.g. "water", "plate", etc). Moreover, 7 prepositions were discovered that describe location, direction or means by which an action is achieved (e.g. "in", "from", "with"). No states were discovered in this example. This is due to the oversimplified sentence structure that follows the pattern "action_direct-object(s)_location(s)".

[2]In our case, that is the actual action executed by the user.

[3]This can be done with the help of parser that POS-tags the text. Later, all present tense verbs are extracted, as they usually describe an action that is executed.

1	Take the knife from the counter.
2	Cut the carrots.
3	Put the knife on the counter.
4	Take the pot from the counter.
5	Put the pot on the stove.
6	Put the carrots in the pot.
7	Turn on the stove.
8	Take the wooden spoon from the counter.
9	Put the wooden spoon in the pot.
10	Cook for 10 minutes.
11	Turn off the stove.
12	Open the cupboard.
13	Take a plate from the cupboard.
14	Take a glass from the cupboard.
15	Put the plate and the glass on the counter.

Table 1: Excerpt from instruction describing the cooking of a carrot soup.

	wash	turn on	turn off	take	sit	put	open	go
1	0	0	0	0	0	0	0	0
2	0	0	0	0	0	1	0	0
3	0	0	0	1	0	0	0	0
4	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0
6	0	0	0	0	0	1	0	0
7	0	0	0	0	0	0	0	0
8	0	0	0	1	0	0	0	0
9	0	0	0	0	0	1	0	0
10	0	0	0	0	0	0	0	0
11	0	0	0	0	0	1	0	0
12	0	1	0	0	0	0	0	0
13	0	0	0	0	0	0	0	0
14	0	0	0	1	0	0	0	0
15	0	0	0	0	0	1	0	0

Table 2: The time series corresponding to the text in Table1.

The next step is to represent each action as a time series. More precisely, each element in the time series is represented with a number. This number indicates the number of occurrences of the given action in the current sentence. Table 1 shows the time series to some of the words extracted from the text in Table 1 In that manner, each of the words (or pairs of words) of interest is assigned a time series. This allows the utilisation of time series analysis for the discovery of implicit causal relations in textual instructions. The resulting time series can be downloaded from (Yordanova 2015b).

Figure 6 shows the causal relations between actions discovered for cooking a carrots soup.

After identifying the causal relations, the argument ontology is learned. This is done by feeding the identified nouns to WordNet in order to build the initial ontology. Then, based on the relations described through prepositions, similar causal relations, and abstraction in the action ontology, the argument ontology is refined and new relations are identified. Figure 7 shows the resulting ontology and the steps for building it. Similarly to the argument ontology, the action ontology is based on the identified actions, the objects they are executed on and the indirect objects or locations where they are executed. Each abstraction level of the action ontology is based on the corresponding abstraction level in the argument ontology.

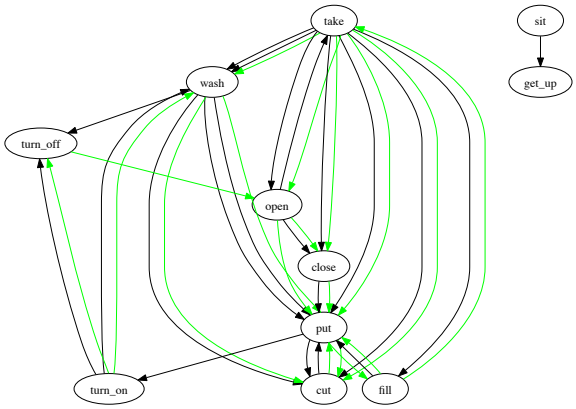

Figure 6: Causal relations discovered for cooking a carrot soup. Black indicates relations discovered by a human annotator, green: discovered with the proposed approach.

In the next step, based on the action ontology and the identified causal relations, the precondition-effect rules are built. In this example, the rules are built based on 17 predicates. As there were no states discovered, the predicates indicate whether an action is executed or not (e.g. "(executed-wash ?f - wash-obj)"). Based on these rules, 20 action templates were constructed. The templates are more than the action classes because the same action class has different preconditions or effects in different situations. Figure 8 shows the generated precondition-effect rule for the action "close". It can be seen that apart from the typical PDDL action notation, there are two additional slots: ":duration" and ":observation". These are later used for performing activity recognition. There, the actions have durations and are observed through sensor observations. These slots allow linking the

```
(:action close
  :parameters (?c - area)
  :duration (closeDuration)
  :precondition (and
    (not (executed-close ?c))
    (executed-open ?c)
  )
  :effect (and
    (executed-close ?c)
    (not (executed-open ?c))
  )
  :observation  (setActivity (activity-id
      close))
)
```

Figure 8: The action template *close* in the PDDL notation.

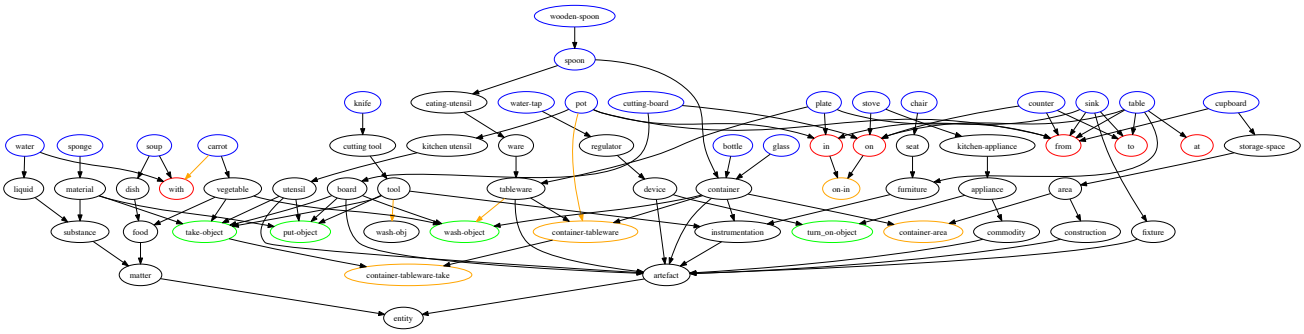

Figure 7: Left: Learning the argument ontology. Step 1 (blue): objects identified through POS-tagging and dependencies; step 2 (black): hierarchy identified through WordNet; step 3 (red): types identified through the relations of objects to prepositions; step 4 (green): types identified based on similar preconditions; step 5 (yellow): types identified through action abstraction. Right: Learning the action ontology: (uppermost layer): abstract representation; (middle layer): abstract representation of action "take"; (bottommost layer): concrete representation of action "take"".

behaviour model to the underlying action duration distribution and the expected type of observations.

After defining the action rules, the next step is to generate the initial and goal states. The initial and goal states represent different combinations of truth values over all ground predicates. In our example we have 204 ground predicates which means that we have 204! possible combinations. To reduce this number, we utilise some prior knowledge. We assume that none of the objects is taken, no doors or cupboards are open, and no devices are turned on. This means that the predicates (executed-close), (executed-put), (executed-turn-off) are set to true. We also assume that apart from these, no other actions have been executed at the beginning of the problem. This leaves us with only one initial state. Similarly, for the goal state we assume that the actions "cook", "drink", "eat", and "wash" (applied to the different objects) have to be executed and for the rest we do not care. As in the experiment we conducted to collect the sensor data, different persons chose to wash different objects, we generate different goal conditions. There, the predicates indicating the execution of the actions "cook", "drink", and "eat" are always set to true, and the truth value of the predicates describing the execution of the "wash" actions vary. As we have 7 objects on which "wash" can be executed, this means we have 5040 combinations of goal conditions.

Having defined the initial state and the goals, we use a state of the art planner to identify any problems in the models that prevent reaching the goal state. For example, the causal relation "fill causes take" was discovered, which means that in the precondition of "take" the predicate (executed-fill) has to be true. However, this prevents the model from reaching the goal state as no objects can be taken unless the action "fill" is executed. For that reason we remove this condition from the precondition of "take". This procedure is repeated for all models until all problems preventing the models from reaching a goal state are removed.

The result of this step is a set of causally correct models that contain all execution sequences from the initial state to the possible goal states. If the model can be fully extended it can be seen as a directed graph where the nodes are states and the transitions are actions. The graph starts with the initial state where the probability of this state is one in the case of only one state, otherwise there is a probability distribution over all initial states. The transitions from one state to another also have probabilities, which are defined based on action selection strategy such as the distance to the goal, or how often the action is executed etc. Figure 9 shows an example of such graph where the dots are the states and the connections between them are the actions. The graph starts with one initial state and then follows different paths to the goal states (red dots at the bottom of the graph). In this case the graph was only partially explored with iteratively deepening depth first search due to the large state space. The red line shows the sequence of actions the user actually executed.

As we now have 5040 candidate models, we use a set of plans describing the execution sequences in the experiment we conducted. In this example the plans are generated from the annotation of the video logs recorded during the experiment. This step reduces the model to one goal condition where apart from "cook", "eat", and "drink", "wash glass", "wash carrot", and "wash plate" have to be executed. For the remaining predicates, we do not care about their truth value[4].

The resulting model has very large branching factor[5]. This reduces the probability of selecting the actual action being executed, especially in the case of noisy or ambiguous observations. For that reason an optimisation step is applied. Using the model and sensor observations describing the execution sequences of different users preparing a carrot soup, the

[4]This one condition generates a set of possible goal states that can be reached. In other words, our model now contains one initial state and several goal states in which the goal condition is satisfied.

[5]This is the maximum number of actions executable from a given state, or in other words the maximum number of connections that leave a dot in Figure 9.

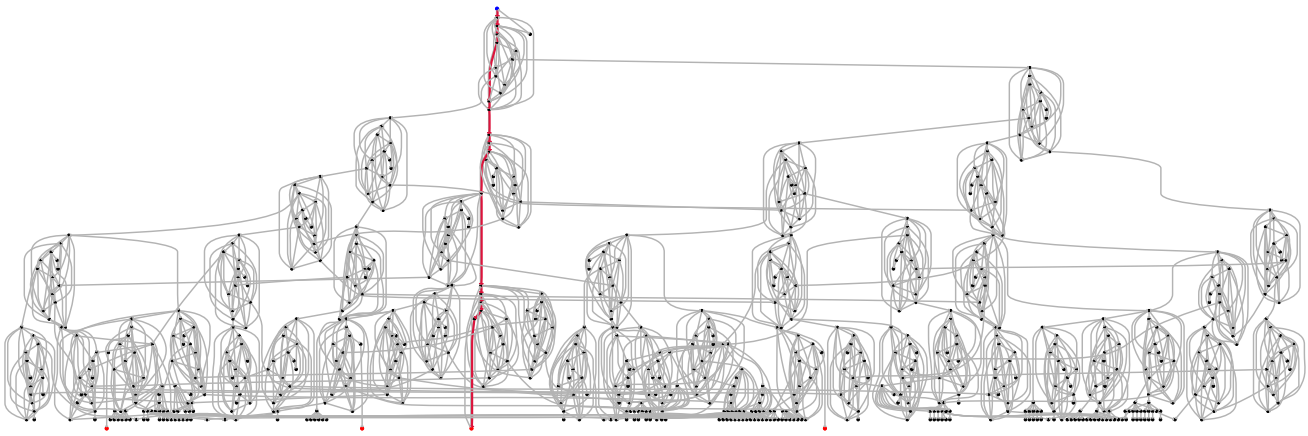

Figure 9: Partially extended state space graph of the model.

transition probabilities are then adjusted and the observed sequences receive higher probability than those that are not observed. In that manner, more typical behaviour is more likely to be observed, but in the same time less probable behaviour is not completely removed, so that in case of new observations, the model can be further adjusted.

Discussion

In this work we proposed an approach that automatically extracts knowledge from textual instructions and learns models of human behaviour that are later used for activity recognition. One problem the approach faces is the discovery of causal relations. As textual instructions such as recipes are usually written in informal manner, their sentences often lack the direct objects. This in turn makes it difficult to discover the objects on which an action is executed and also reduces the ability of the approach to discover causal relations. For that reason, we believe that the approach could benefit from anaphora resolution techniques in order to include the missing direct objects to the sentence.

Another problem is the generation of initial and goal states. As it could be seen from the example, even with simplifying assumptions, there is a very large number of possible combinations. In that sense, the approach could benefit of automated techniques that discover improbable initial and goal conditions in the text. This could potentially reduce the number of candidate models thus reducing the computational effort required to evaluate the applicability of the models to the problem at hand.

Finally, the approach could potentially benefit from utilising multiple textual instructions to learn the candidate models. This will allow the generation of richer models that contain more contextual information and that are not tailored for only one specific case.

Current Results and Future Work

In a previous work we proposed an approach of extracting causal relations from textual instructions through time series analysis (Yordanova 2015a). We applied the approach to 20 textual instructions (cooking recipes, manuals, and experiment instructions). The results showed that the approach is able to identify causal relations in simple texts with short sentences. We compared the approach to one based on grammatical patterns and discovered that the latter was able to detect very low number of relations. We used the extracted relations as a basis for building precondition-effect rules (Yordanova and Kirste 2016). In (Yordanova and Kirste 2016) we were able to learn a causal model describing the activities from the carrots soup preparation dataset and to compare it to a manually built model described in (Yordanova and Kirste 2015). The results showed that our approach is able to extract precondition-effect rules and to explain the plans corresponding to the video logs from the dataset. They, however, showed that the action probability of executing the action described in the plan is very low given the model. It also has to be mentioned, that the initial and goal state of the model were manually defined.

In the future we will concentrate on optimising the textual instructions by applying anaphora resolution techniques. We will also investigate techniques for reducing the set of possible initial and goal states before the optimisation step. Furthermore, we will investigate inverse reinforcement learning methods for optimising the model structure from sparse observations. Finally, we plan to apply the approach to different domains (such as physiotherapy and assistance of workers) in order to test its applicability to various problems from the domain of daily activities.

If successful, the proposed approach will reduce the need of expert knowledge by replacing it with the domain knowledge encoded in textual instructions. It, in turn, will reduce the time and resources needed for developing computational models of human behaviour for activity recognition. It will also be the first attempt at actually applying CSSMs learned from textual data to an activity recognition problem.

Acknowledgments

This work is funded by the German Research Foundation (DFG) within the context of the project TextToHBM, grant

number YO 226/1-1.

References

Babeş-Vroman, M.; MacGlashan, J.; Gao, R.; Winner, K.; Adjogah, R.; desJardins, M.; Littman, M.; and Muresan, S. 2012. Learning to interpret natural language instructions. In *Proceedings of the Second Workshop on Semantic Interpretation in an Actionable Context*, SIAC '12, 1–6. Stroudsburg, PA, USA: Association for Computational Linguistics.

Branavan, S. R. K.; Kushman, N.; Lei, T.; and Barzilay, R. 2012. Learning high-level planning from text. In *Proc. of the Annual Meeting of the Association for Computational Linguistics*, 126–135.

Branavan, S. R. K.; Silver, D.; and Barzilay, R. 2011. Learning to win by reading manuals in a monte-carlo framework. In *Proc. of the Annual Meeting of the Association for Computational Linguistics*, 268–277.

Branavan, S. R. K.; Zettlemoyer, L. S.; and Barzilay, R. 2010. Reading between the lines: Learning to map high-level instructions to commands. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, ACL '10, 1268–1277. Stroudsburg, PA, USA: Association for Computational Linguistics.

Chen, D. L., and Mooney, R. J. 2011. Learning to interpret natural language navigation instructions from observations. In *Proc. of the AAAI Conference on Artificial Intelligence*, 859–865.

Chen, L.; Hoey, J.; Nugent, C.; Cook, D.; and Yu, Z. 2012. Sensor-based activity recognition. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews* 42(6):790–808.

Goldwasser, D., and Roth, D. 2014. Learning from natural instructions. *Machine Learning* 94(2):205–232.

Granger, C. W. J. 1969. Investigating Causal Relations by Econometric Models and Cross-spectral Methods. *Econometrica* 37(3):424–438.

Hiatt, L. M.; Harrison, A. M.; and Trafton, J. G. 2011. Accommodating human variability in human-robot teams through theory of mind. In *Proc. of the Int. Joint Conference on Artificial Intelligence*, 2066–2071.

Kollar, T.; Tellex, S.; Roy, D.; and Roy, N. 2014. Grounding verbs of motion in natural language commands to robots. In Khatib, O.; Kumar, V.; and Sukhatme, G., eds., *Experimental Robotics*, volume 79 of *Springer Tracts in Advanced Robotics*. Springer Berlin Heidelberg. 31–47.

Krüger, F.; Nyolt, M.; Yordanova, K.; Hein, A.; and Kirste, T. 2014. Computational state space models for activity and intention recognition. a feasibility study. *PLoS ONE* 9(11):e109381.

Krüger, F.; Hein, A.; Yordanova, K.; and Kirste, T. 2015. Recognising the actions during cooking task (cooking task dataset). University Library, University of Rostock. http://purl.uni-rostock.de/rosdok/id00000116.

Li, X.; Mao, W.; Zeng, D.; and Wang, F.-Y. 2010. Automatic construction of domain theory for attack planning. In *Int. Conference on Intelligence and Security Informatics*, 65–70.

Miller, G. A. 1995. Wordnet: A lexical database for english. *Commun. ACM* 38(11):39–41.

Nguyen, T. A.; Kambhampati, S.; and Do, M. 2013. Synthesizing robust plans under incomplete domain models. In *Advances in Neural Information Processing Systems*. Curran Associates, Inc. 2472–2480.

Philipose, M.; Fishkin, K. P.; Perkowitz, M.; Patterson, D. J.; Fox, D.; Kautz, H.; and Hahnel, D. 2004. Inferring activities from interactions with objects. *IEEE Pervasive Computing* 3(4):50–57.

Ramirez, M., and Geffner, H. 2011. Goal recognition over pomdps: Inferring the intention of a pomdp agent. In *Proc. of the Int. Joint Conference on Artificial Intelligence*, volume 3, 2009–2014.

Sil, A., and Yates, E. 2011. Extracting strips representations of actions and events. In *Recent Advances in Natural Language Processing*, 1–8.

Tenorth, M.; Nyga, D.; and Beetz, M. 2010. Understanding and executing instructions for everyday manipulation tasks from the world wide web. In *Int. Conference on Robotics and Automation*, 1486–1491.

Ye, J.; Stevenson, G.; and Dobson, S. 2014. Usmart: An unsupervised semantic mining activity recognition technique. *ACM Transactions on Interactive Intelligent Systems* 4(4):16:1–16:27.

Yordanova, K., and Kirste, T. 2015. A process for systematic development of symbolic models for activity recognition. *ACM Transactions on Interactive Intelligent Systems* 5(4).

Yordanova, K., and Kirste, T. 2016. Learning models of human behaviour from textual instructions. In *Proceedings of the 8th International Conference on Agents and Artificial Intelligence (ICAART 2016)*, 415–422.

Yordanova, K. 2015a. Discovering causal relations in textual instructions. In *Recent Advances in Natural Language Processing*, 714–720.

Yordanova, K. 2015b. Time series from textual instructions for causal relations discovery (causal relations dataset). University Library, University of Rostock. http://purl.uni-rostock.de/rosdok/id00000117.

Yordanova, K. 2016. From textual instructions to sensor-based recognition of user behaviour. In *Companion Publication of the 21st International Conference on Intelligent User Interfaces*, IUI '16 Companion, 67–73. New York, NY, USA: ACM.

Zhang, Z.; Webster, P.; Uren, V.; Varga, A.; and Ciravegna, F. 2012. Automatically extracting procedural knowledge from instructional texts using natural language processing. In *Proc. of the Int. Conference on Language Resources and Evaluation*.

Zhuo, H. H., and Kambhampati, S. 2013. Action-model acquisition from noisy plan traces. In *Proc. of the Int. Joint Conference on Artificial Intelligence*, 2444–2450.