Problem Solving Using Classical Planners
AAAI Technical Report WS-12-12

# Learning Interactions Among Objects Through Spatio-Temporal Reasoning

**Mustafa Ersen** and **Sanem Sariel-Talay**
Artificial Intelligence and Robotics Laboratory
Computer Engineering Department
Istanbul Technical University, Istanbul, Turkey
{ersenm,sariel}@itu.edu.tr

## Abstract

In this study, we propose a method for learning interactions among different types of objects to devise new plans using these objects. Learning is accomplished by observing a given sequence of events with their timestamps and using spatial information on the initial state of the objects in the environment. We assume that no intermediate state information is available about the states of objects. We have used the Incredible Machine game as a suitable domain for analyzing and learning object interactions. When a knowledge base about relations among objects is provided, interactions to devise new plans are learned to a desired extent. Moreover, using spatial information of objects or temporal information of events makes it feasible to learn the conditional effects of objects on each other. Our analyses show that, integrating spatial and temporal data in a spatio-temporal learning approach gives closer results to that of the knowledge-based approach by providing applicable event models for planning. This is promising because gathering spatio-temporal information does not require great amount of knowledge.

## Introduction

Automated action planning is the process of finding a sequence of actions to achieve a given goal. Whenever the planning problem (i.e., initial and goal states) and operators corresponding to domain actions are defined properly, a plan to solve the problem can be devised by a planner. However, the problem gets complicated when the planning agent has incomplete information about the preconditions and the effects of the domain operators. In this case, the problem turns into a learning task where the relations among actions and objects are learned through the observations.

The main goal of this research is to develop a method for learning object interactions through observation of a given sequence of events. We analyze the performance of the learner against the level of completeness of the knowledge base about relations. We believe the outcomes of this study are also useful for robot learning tasks. Our future work includes implementation of this system on real robots for learning affordances.



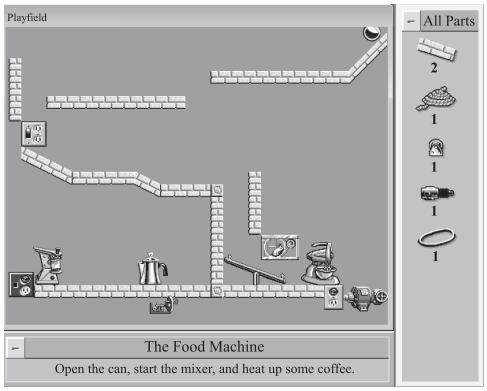



Figure 1: An example puzzle from TIM v3.

We have selected the Incredible Machine (TIM) as a domain for analyzing these interactions because this domain allows the use of various objects and their interactions to build complex systems to solve given puzzles. TIM is a series of computer games in which the player tries to solve given puzzles by constructing contraptions in cartoonist Rube Goldberg's style (Wolfe 2011). In a typical scenario of this game, the player is given a limited number of objects, tools and machines and a goal to accomplish by using these resources. An example screenshot is provided in Fig. 1 to illustrate the game environment. In this example, the aim is opening the can, starting the mixer and heating up some coffee. To complete the given structure to reach the goal state, the player is provided with some objects in the parts bin on the right side, and these parts must be used without changing the positions of existing objects in the environment.

This problem fits into planning framework as it involves events to transform a given initial state into a goal state. Both progression and regression planning is applicable if domain events are perfectly defined. A planner needs to consider the game in two aspects to be able to devise a consistent plan. First, considering interactions among objects, tools and machines is crucial for making decisions while connecting given parts altogether to reach the goal state. Second, physics models of objects (e.g., the effect of the gravity on the motion of a ball) should be taken into account to estimate interactions through motions.

In this work, we focus on solving the stated learning problem where preconditions and effects of events in the domain are not known but interactions of objects are given in a tutorial to understand the rationale behind TIM puzzles. We assume that a given tutorial presents object interactions in the form of chain reactions (e.g., starting a mixer, running a motor, lighting a lamp, etc.). No further background information is available about the types of objects and their semantics, nor intermediate state information. Particularly in this work, tutorials are presented as text-based event sequences. Extension of the work by processing visual observations as in the actual game is left as a future work. We investigate the problem from four perspectives and propose an integrated solution. If relations of parts are provided to the agent, a logic-based reasoning mechanism could be used. Otherwise, we show that considering spatial and temporal aspects of the problem makes it possible to reason about interactions for further planning tasks. These perspectives may correspond to different learning problems. TIM domain provides a suitable framework to integrate these approaches.

The rest of the paper is organized as follows. First, we review earlier work in the field of learning action schema. Next, we give an overview of our system and show the details step by step on a running example. Third, we show how to use learned information on a planning example. Finally, we present experiments on the proposed system, discuss the results and conclude the paper.

## Background

In recent years, efficient approaches have been developed to learn planning operators from example plans. Among these, SLAF (Simultaneous Learning and Filtering) is a tractable system for identifying preconditions and effects of actions (Shahaf 2007). It builds exact action schema efficiently by using a relational logical representation and uses partial observations to identify consistent action models. However, this approach requires the use of observations on the states of objects between actions.

Unlike SLAF, ARMS (Action-Relation Modelling System) can learn action models without intermediate state information (Wu, Yang, and Jiang 2007). ARMS is based on solving a weighted MAX-SAT problem on the statistical distribution of related action and predicate sets. It is an efficient approach to find approximate STRIPS action schema by using only actions and predicates in the plan trace. However, this approach cannot learn actions with conditional effects for identifying interactions among objects, tools and machines as in our problem. LAMP (Learning Action Models from Plan traces), learns more expressive action schema having quantifiers and logical implications (Zhuo et al. 2010). This approach is based on enumerating six different patterns of candidate formulas and their quantified cases on training data to learn preconditions, effects and conditional effects. In LAMP, partial observations between action executions are crucial for efficiency.

OPMAKER (Richardson 2008) is another operator induction tool which can learn domain operators from plan traces. In addition to plan traces, to learn the preconditions and effects of actions, OPMAKER requires the information on initial and intermediate states of the environment and a partial domain model to be provided. OPMAKER2 (Richardson 2008) is an extension which can induce action schema from training examples without using intermediate state information. However, this method also requires a partial domain model including information on the object types.

LOCM (Learning Object-Centred Models) is an object-centered approach for learning action models from plan traces (Cresswell, McCluskey, and West 2009). It's unique in its input as it only requires the action sequence to learn action schema. Moreover, LOCM outputs finite state machines (FSM) showing behaviors of different types of objects in the domain, as well as planning operators. LOCM2 (Cresswell and Gregory 2011) is proposed to solve the state generalization problem of LOCM for objects showing behaviors in separate aspects. By analyzing example plans in a transition-based representation, LOCM2 weakens the assumptions of LOCM and produces additional state machines to capture behaviors that cannot be modeled by LOCM. For our work, LOCM is the most suitable base system to determine the behaviors of the objects as it generates FSMs in a state-based representation. We extend LOCM in two ways. First, we include different orientations of the same object in input to distinguish the change of behaviors due to the changes in orientations. Second, we propose an approach to determine interactions among objects when relational information is given. Our approach uses a forward chaining mechanism to produce conditional effects in a chain reaction. Furthermore, we analyze the use of spatial and temporal information in case a knowledge base of relations is not available.

## Learning Interactions from Examples

Learning is essential for an agent with limited knowledge about its environment. The problem that we investigate in this work asks for learning interactions of events and objects for TIM game, given some tutorials about this game. Tutorials, represented in a text-based format, include sequential events on various objects. Aside from these tutorials, there exists no prior knowledge about events, their preconditions and effects, objects and their interactions.

A tutorial is a tuple $T = (\mathcal{E}, \mathcal{F}, \mathcal{R})$ where $\mathcal{E}$ is an event sequence with $n$ events $E_{1:n}$ in their order of occurrence where concurrent events $(\forall i, j\ i < j \Rightarrow E_i \preceq E_j)$ affect different objects, $\mathcal{F}$ is a list of predicates showing different orientational features of objects (e.g., $facing\_left$ or $facing\_right$) in the environment and $\mathcal{R}$ is a list of relations among objects (i.e., a knowledge base of directly observable relations or an automatically constructed list by using spatial locality of objects or temporal locality of events). Each event $E_i$ has $m_i$ arguments where each argument is an object in the domain $(Arg_{1:m_i}(E_i) = O_{i,k}, 1 \leq k \leq m_i)$.

The main objective is to model behaviors of different types of objects represented as FSMs. Each FSM involves a state set $\mathcal{S}$ and a state-transition function $\delta$. Furthermore, conditional relations among these FSMs should be established by using $\mathcal{R}$ to capture interactions among different types of objects and determine preconditions of events given in $\mathcal{E}$. Eventually, these preconditions together with behavior models are to be used for generating new plans.

## The Proposed System

We propose a two-phase solution to the learning problem. In the first phase, objects are grouped into sets with respect to their type; and a FSM is generated for each set to model behaviors of objects belonging to that set. State machines are created using the steps of LOCM. We propose a modification to address interactions based on different orientational features of objects. In the second phase, interactions among objects from different sets are determined by establishing conditional links between related states. Connections and relations among objects are used while generating these conditional structures to model interactions.

Throughout the paper we illustrate our system on an example tutorial whose initial state is visualized in Fig. 2. This example tutorial includes various objects and interactions. The objects presented in the tutorial (except the balls) are not affected by the gravity and motion laws. In this scenario, after dropped from a certain height, $ball_1$ turns on $switch_1$ which initiates two concurrent chains of reactions. The event sequence given on the right side represents this tutorial in a text-based format.

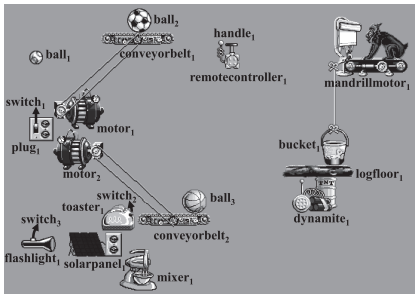



Figure 2: An example scenario including interactions among various objects. $motor_1$, $motor_2$, $mixer_1$ and $toaster_1$ require electricity to work. $motor_1$ and $motor_2$ are plugged to $plug_1$ which is controlled by $switch_1$. $toaster_1$ and $mixer_1$ are plugged to $solarpanel_1$ which can generate electricity from light. $flashlight_1$ is directed at $solarpanel_1$. To make toast, $switch_2$ on $toaster_1$ needs to be pushed. $motor_1$ drives $conveyorbelt_1$ by a belt and similarly $motor_2$ drives $conveyorbelt_2$. $dynamite_1$ is remotely controlled by $remotecontroller_1$. $logfloor_1$ dissolves when $dynamite_1$ explodes. The monkey runs $mandrillmotor_1$ if the shade covering bananas is pulled by a rope.

### Phase-1: Creating FSMs Reflecting Behaviors

In the first phase, objects are grouped with respect to their type; and behaviors are modeled by using a unique FSM for each group of objects. As our system initially applies the steps given in LOCM, we first repeat the existing underlying assumptions here, for convenience. Then, we present our extension to model different features of objects.

LOCM groups objects into sets called *sorts*; and the behavior of each *sort* is modeled by using a single state machine per *sort*. FSMs are created based on the following assumptions made by LOCM.

1. Each argument of the same type of action is restricted to contain objects of the same *sort*.
2. Each action causes a transition on each object in its arguments. Some transitions may not change the state of an object.
3. The same transition cannot appear more than once in a FSM belonging to the *sort* it affects.

By using Assumption 1, LOCM divides objects appearing in action parameters (or event parameters) into a set structure called *sorts* where each *sort* is a disjoint subset of all the objects in the domain. If an event name $name(E)$ with parameters $p$ appears in both $i^{th}$ and $j^{th}$ entries in the event sequence $\mathcal{E}$, the corresponding objects $O_{i,k}$ and $O_{j,k}$ for each parameter $1 \leq k \leq p$ of events $E_i$ and $E_j$ are considered to share the same *sort*. In this work, we use this assumption to determine the type of each object. When we consider $ball_1$, $motor_1$, $motor_2$ and $plug_1$ objects in Fig. 2, after $ball_1$ pushes $switch_1$ of $plug_1$ down, both $motor_1$ and $motor_2$ start running. This is given in the event sequence $\mathcal{E}$ as events $E_{1:3}$. In this sequence $name(E_2) = name(E_3) = start$, thus the objects in $Arg_1(E_2) = O_{2,1}$ and $Arg_1(E_3) = O_{3,1}$ are grouped into the same *sort* as $\{motor_1, motor_2\}$. Grouping is done similarly for the other related objects in the domain.

After grouping objects into *sorts*, LOCM uses the input action sequence to generate FSMs that model the behavior of each *sort*. Similarly in our problem, each event $E_i \in \mathcal{E}$ is assumed to cause a transition for each object $k$ in its arguments where $1 \leq k \leq p$ (Assumption 2). Each transition $E_{i,k}$ has a start and an end state, namely $start(E_{i,k})$ and $end(E_{i,k})$. The end state of a transition could be the same as its start state. All transitions are assumed to be one-to-one, thus $start(E_{i,k}) = start(E_{j,k})$ and $end(E_{i,k}) = end(E_{j,k})$ for each occurrence of the same event $E$ in $\mathcal{E}$ ($E = name(E_i) = name(E_j)$). After assigning a start and an end state for each transition in the action sequence, LOCM utilizes the continuity of transitions on the same object instance to determine equivalent states in the FSM of each *sort*. Transitions $E_{i,k}$ and $E_{j,l}$ are said to be consecutive with respect to an object $O$ iff $O = O_{i,k} = O_{j,l}$, and $O$ does not occur in the parameters of event instances between $i$ and $j$. To illustrate how this assumption is used, consider $ball_1$ object in the example event sequence. This object is included in the first parameter of both $E_1 = push\_down$ and $E_8 = press$. In the beginning, the corresponding transitions are initialized as,

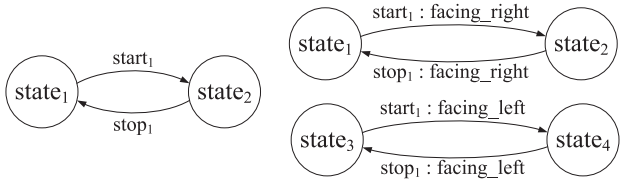
$$state_{1,1} \xrightarrow{push\_down_1} state_{2,1}$$
$$state_{3,1} \xrightarrow{press_1} state_{4,1}$$

By using the continuity of transitions, the end state of $push\_down_1$ and the start state of $press_1$ are unified as

$$state_{1,1} \xrightarrow{push\_down_1} state_{2,1}$$
$$state_{2,1} \xrightarrow{press_1} state_{4,1}$$

where $state_{i,j}$ means the $i^{th}$ state of the $j^{th}$ *sort* in the training set and the *sort* identifier of $ball_1$ is assigned 1 as it is the first object in the event sequence.

The basic steps of LOCM are useful to create corresponding state machines. However, modeling behavior changes due to different orientations defined in $\mathcal{F}$ is not available. For example, $motor_1$ and $motor_2$ are instances from the same *sort*, but they are placed in opposite directions. As a result, $conveyorbelt_1$ and $conveyorbelt_2$ (which are connected to $motor_1$ and $motor_2$ by using belts, respectively) rotate in opposite directions. For the motor *sort*, LOCM produces the FSM given in Fig. 3 (a). This FSM does not present the difference between these rotations. Neither LOCM2 features this property because event names are the same in both cases.




(a) FSM produced by LOCM (b) FSMs produced by our system

Figure 3: Resulting FSMs for the motor *sort*.

We have extended LOCM to deal with these issues. When the objects are placed in opposite directions, their effects may be different as for the two motors given in Fig. 2. If their features are stated as reflecting a change in their orientations, our proposed extension can detect the difference in effects. When the input includes the following facts: $facing\_left(motor_1)$ and $facing\_right(motor_2)$, the desired output is formed as in Fig. 3 (b). This is achieved by considering feature differences in the FSMs. We assume that each different feature of the same object causes a change in the effects of transitions on this object. For that reason, in our approach, transitions on objects in changing forms are assigned different start and end states. Therefore, two FSMs are created for the motor *sort*. $state_1$ and $state_3$ are different from each other as the same motor object cannot change its orientation between $facing\_right$ and $facing\_left$ during runtime. Similarly, the end states are different.

## Phase-2: Modeling Interactions through Relations

In the second phase of our system, connections among objects from different *sorts* are used to determine and model interactions among objects through events. We analyze four different input types. First, we investigate the case where background knowledge on relations is given. In this case, a forward chaining approach could exploit these relations to make conclusions. In the second and third analyses, we investigate how the results change if spatial locality of objects or temporal locality of events is taken into account without prior domain knowledge. Finally, we analyze the use of spatial and temporal information together to infer interactions among objects, tools and machines.

**Using a Knowledge-Based Representation** To capture interactions among different state machines, first we encoded a knowledge base consisting of predicates representing directly observable connections among objects for a given tutorial. Our system uses these predicates to reason about interactions among objects. These predicates are shown in Table 1 along with their direction of connection and some examples. Note that the knowledge base does not contain semantic information about the types of objects.

| Predicate | Examples from Fig. 2 | Direction |
|---|---|---|
| $plugged(obj_1, obj_2)$ | $plugged(motor_1, plug_1)$<br>$plugged(motor_2, plug_1)$<br>$plugged(mixer_1, solarpanel_1)$<br>$plugged(toaster_1, solarpanel_1)$ | $obj_2 \rightarrow obj_1$ |
| $on(obj_1, obj_2)$ | $on(ball_2, conveyorbelt_1)$<br>$on(ball_3, conveyorbelt_2)$<br>$on(bucket_1, logfloor_1)$ | |
| $belt(obj_1, obj_2)$ | $belt(motor_1, conveyorbelt_1)$<br>$belt(motor_2, conveyorbelt_2)$ | $obj_1 \rightarrow obj_2$ |
| $facing(obj_1, obj_2)$ | $facing(flashlight_1, solarpanel_1)$ | |
| $has(obj_1, obj_2)$ | $has(plug_1, switch_1)$<br>$has(toaster_1, switch_2)$<br>$has(flashlight_1, switch_3)$<br>$has(remotecontroller_1, handle_1)$ | $obj_1 \leftrightarrow obj_2$ |
| $rope(obj_1, obj_2)$ | $rope(mandrillmotor_1, bucket_1)$ | |
| $near(obj_1, obj_2)$ | $near(dynamite_1, logfloor_1)$ | |

Table 1: Types of Connections Among Objects

When these facts are provided, our system applies a forward chaining mechanism to create a list of connections for each object in the domain. For example, there are two applicable connections for $toaster_1$ in Fig. 2: $has(toaster_1, switch_2)$, $plugged(toaster_1, solarpanel_1)$. Thus, the forward chaining mechanism creates two subsets of connections for $toaster_1$ as $\{[switch_2], [solarpanel_1]\}$ in the first iteration. The future iterations of forward chaining use transitivity of these connections to determine chains of relations. Considering new relations in the chain, the connection set for $toaster_1$ is extended to $\{[switch_2], [solarpanel_1, flashlight_1, switch_3]\}$. The objects that are not represented in the event sequence $\mathcal{E}$ are removed from the list of connections. For example, objects $solarpanel_1$, $plug_1$ and $logfloor_1$ are removed from the final connection sets. Therefore, the final connection set for $toaster_1$ becomes $\{[switch_2], [flashlight_1, switch_3]\}$ after these eliminations.

Finalized connection sets for all objects are used in a process to determine interactions among objects through events given in $\mathcal{E}$. It is important to determine the preconditions as a complete set of facts to execute an action given a chain of reactions. However, this is not completely possible because state information (i.e., states of objects) is not available as an input. The intuitive idea that we apply here includes the use of the last effect for each subset of the resulting connection set. Therefore, consecutive transitions among connected objects are extracted from $\mathcal{E}$ by using this intuition. For example, the preconditions of $E_{15} : make\_toast(toaster_1)$ can be finalized by using this approach on the resulting connection set $\{[switch_2], [flashlight_1, switch_3]\}$. The last observed event on $switch_2$ is $E_{11} : push(ball_3, switch_2)$ and the last observed event on the second subset is $E_9 : light(flashlight_1)$. Hence, the preconditions of

$make\_toast$ include both $push$ and $light$ events under the conditions $plugged(toaster_1, solarpanel_1)$, $facing(flashlight_1, solarpanel_1)$ and $has(toaster_1, switch_2)$.

**Using Spatial Locality of Objects** When a knowledge base presenting connection types is not provided, the only input to the system is the event sequence. Human players are given the visual scene of objects from which interactions should be determined. An automated agent should also extract the relevant location information of objects to reason correctly. Computer vision techniques are useful for this purpose. The approach that we consider here is using 2D template matching techniques (Brunelli 2009) followed by fitting Minimum Bounding Rectangles (MBR) for objects to capture spatial information (Wood 2008). We assume that location information for each object is provided including its horizontal and vertical dimensions in pixels and the coordinate of the left-top corner. By using this information, three kinds of predicates are extracted: $near$, $tangent$ and $has$. As shown in Fig. 4 (a), this is done by reasoning as,

$$0 < distance(rec_i, rec_j) < \varepsilon \Rightarrow near(object_i, object_j)$$
$$distance(rec_i, rec_j) = 0 \Rightarrow tangent(object_i, object_j)$$
$$rec_j \subset rec_i \Rightarrow has(object_i, object_j)$$

where $rec_i$ and $rec_j$ are used for the minimum bounding rectangles of $object_i$ and $object_j$, respectively.

In this model, $tangent$ predicate is similar to *externally connected* relation in RCC8 (Randell, Cui, and Cohn 1992) and $has$ predicate is similar to *proper part* relation in RCC8. Apart from these, $near$ is a quantitative relation based on distance. In addition to these mereotopological relations, cardinal directions (Frank 1991) are used in spatial reasoning as shown in Fig. 4 (b). Specifying relative object positions makes sense for some of the interactions (e.g., for $on(ball_2, conveyorbelt_1)$, but not for $plugged(motor_1, plug_1)$). The effect of relative positions can be learned by considering cardinal directions.

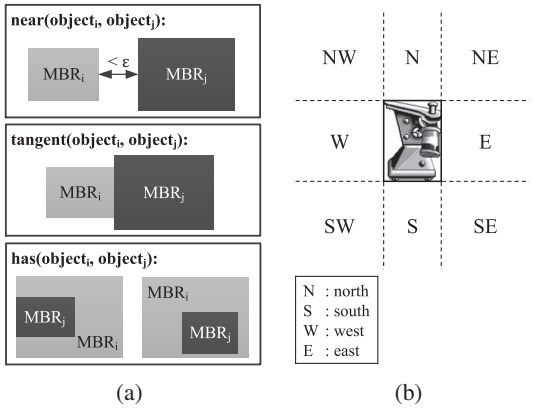



Figure 4: (a) Spatial relation types between MBRs of objects (b) Cardinal directions around MBR of $canopener$ object.

Spatial locality information is useful for extracting spatial relations. However, connections among remotely located objects cannot be captured (e.g., belt, rope and remote controlled dynamite).

**Using Temporal Locality of Events** As spatial reasoning does not cover all type of relations and has some drawbacks, we also analyze the results using temporal locality. In this analysis, there is no knowledge base representing relations and spatial information. Instead, timing (start time) of each event $E_i \in \mathcal{E}$ is included in the input event sequence $\mathcal{E}$. If an event $E_i$ occurs after another event $E_j$ in a small interval, we assume that there is a relation between objects $O_{i,1:m_i}$ and $O_{j,1:m_j}$. This is similar to *begins* relation in Vilain's Point-Interval Algebra (Vilain 1982). Events $E_1$, $E_2$, $E_4$, $E_6$, $E_{12}$, $E_{13}$ and $E_{14}$ in our example, form a chain reaction as illustrated in Fig. 5.

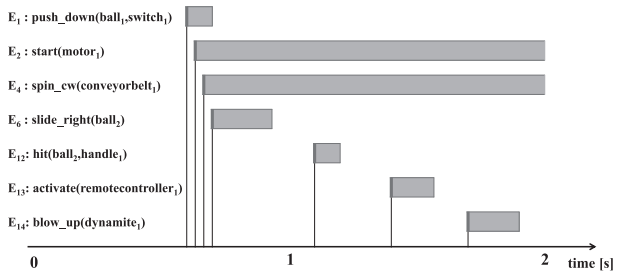



Figure 5: Start time of events in a chain reaction. An example related event pair: $begins(E_{13}, E_{14})$.

The temporal analysis can find several useful relationships for remotely located objects (e.g., $remotecontroller_1$ and $dynamite_1$) when there is a temporal relation. However, this approach suffers mainly from credit assignment problem of choosing the related parameters in an event pair satisfying *begins* relation (e.g., $begins(push\_down(ball_1, switch_1), start(motor_1))$. Furthermore, objects activated by more than one event cannot be modeled. For example, $toaster_1$ runs when both electricity is provided and $switch_2$ is pushed.

**Spatio-Temporal Reasoning** Integration of spatial and temporal information takes advantages of both approaches. Usually the temporal analysis gives better results, although for certain cases using spatial information is superior. These cases are listed below:

- If any event satisfying *begins* pattern has more than one argument (e.g., $push\_down(ball_1, switch_1)$) and the spatial analysis generates a connection (e.g., $switch_1$) including one of these arguments.
- If more than one spatial connection is detected for an object (e.g., $toaster_1$).

In this approach, cardinal direction calculus is used for modeling relative positions of interacting objects. This procedure gives close results as using knowledge base for connections. However, concurrent independent chain reactions cannot be captured by using the integrated reasoner.

## Planning Using Learned Interactions

To show the applicability of the learning results for planning, we use the example scenario illustrated in Fig. 6 (a). In this example, the aim is positioning and connecting given objects in an appropriate way to make a chain reaction which will cause $ball_1$ to fall into the basket ($get\_in(ball_1, basket_1)$). The chain of possible events involving the given objects and necessary relations are shown in the resulting plan given in Fig. 6 (c). These relations must be satisfied as in Fig. 6 (b) in order to reach the goal state. Here, $related$ is a general relation between two objects. Depending on the input type (knowledge base, temporal/spatial information), it could be represented as a corresponding relation (e.g., $has$, $tangent$, $plugged$, etc.). When applicable domain actions (i.e., *place* an object to a position, *flip* the orientation of an object, *connect_with_belt* and *connect_with_rope* to connect two applicable objects using belt or rope, respectively) for a human player is provided, our system can devise a plan to construct the solution in Fig. 6 (b).

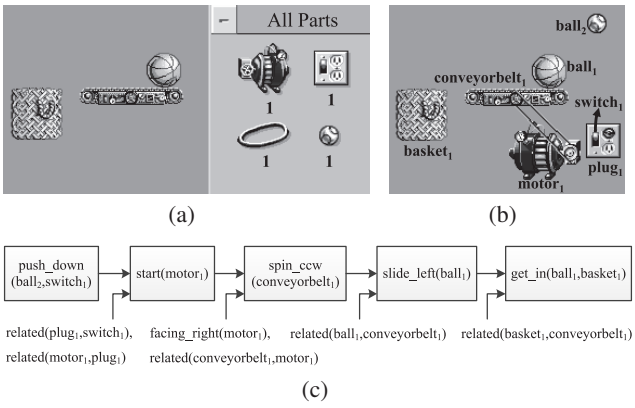



Figure 6: (a) A sample puzzle (b) The solution of the puzzle (c) The plan constructed by using the learned interactions.

## Experimental Evaluation

To evaluate our system, we have created 25 tutorials in TIM domain involving interactions among 40 different object types. In our experiments, we analyze the results from the presented four approaches to learn interactions. Four performance metrics are considered: *accuracy*, *precision*, *recall* and *F-score*. For each object $O_i$ in each tutorial, interactions are classified as *true positives*, *true negatives*, *false positives* and *false negatives* in the following way:

- *True positive*: an existing valid interaction affecting $O_i$ is found correctly.
- *True negative*: no nonexistent interaction is found for $O_i$.
- *False positive*: an invalid interaction is found for $O_i$.
- *False negative*: an existing valid interaction affecting $O_i$ cannot be found.

The results are shown in Table 2. As expected, the knowledge-based approach is superior to all the other approaches, although it is inaccurate in some tutorials involving non-directly observable connections (e.g., $dynamite_1$ and $remotecontroller_1$). The spatial approach suffers from *false positives* due to close proximity of unrelated objects and *false negatives* due to remotely interacting objects. The temporal approach has *false positives* mostly because of choosing related event parameters incorrectly. The spatio-temporal approach is more successful at choosing related event parameters than the temporal approach. Some of the *false positives* in both approaches are due to concurrent independent chain reactions. In addition to the results in Table 2, it has been observed that, using orientational features ($\mathcal{F}$) improves the overall performance in $23.75\%$ of examples. When both *accuracy* and *F-score* values are considered, the spatio-temporal approach is shown to give close results to the knowledge-based approach. These results prove that interactions in the environment can be learned to a great extent by using only spatial and temporal information without any prior knowledge on the part relations.

| Input Type | Accuracy | Precision | Recall | F-Score |
|---|---|---|---|---|
| knowledge base | $98.75\%$ | $100\%$ | $98.10\%$ | $99.04\%$ |
| spatial info. | $71.25\%$ | $76.19\%$ | $71.11\%$ | $73.56\%$ |
| temporal info. | $78.13\%$ | $69.30\%$ | $100\%$ | $81.87\%$ |
| spatio-temporal info. | $91.88\%$ | $88.29\%$ | $100\%$ | $93.78\%$ |

Table 2: Overall Results

We have also analyzed the plans generated by using the learned interactions. In this analysis, the generated plans for 12 TIM puzzles were evaluated with respect to the number of correctly chosen domain actions: *place*, *flip*, *connect_with_belt* and *connect_with_rope*. $91.80\%$ of selected actions by considering learned object models in the knowledge-based approach are correct while the spatio-temporal approach gives close results with $83.61\%$ accuracy.

## Conclusion

We have presented how interactions among objects are learned by using a given sequence of events without semantic information on objects, tools and machines. These learned interactions can later be used by a planner for solving new puzzles in the Incredible Machine domain. Our analysis shows that when a knowledge base about relations is provided, interactions are learned to devise new plans. Our analysis also indicates that the spatial and the temporal approaches have some drawbacks when applied separately. An integrated approach outperforms these two approaches. We have shown that the integrated approach gives close results to that of the knowledge-based approach. This is promising because the spatio-temporal analysis does not require great amount of knowledge on relations. Furthermore, the system can solve puzzles created in the game where only a visual input is provided to the user. Currently, inputs are provided in a text-based event format. Integrating computer vision techniques is left as a future work.

## References


Brunelli, R. 2009. *Template Matching Techniques in Computer Vision: Theory and Practice*. Wiley.

Cresswell, S., and Gregory, P. 2011. Generalised domain model acquisition from action traces. In *Proceedings of the 21st International Conference on Automated Planning and Scheduling (ICAPS'11)*, 42–49.

Cresswell, S.; McCluskey, T. L.; and West, M. M. 2009. Acquisition of object-centred domain models from planning examples. In *Proceedings of the 19th International Conference on Automated Planning and Scheduling (ICAPS'09)*, 338–341.

Frank, A. U. 1991. Qualitative spatial reasoning with cardinal directions. In *Proceedings of the 7th Austrian Conference on Artificial Intelligence*, 157–167.

Randell, D. A.; Cui, Z.; and Cohn, A. G. 1992. A spatial logic based on regions and connection. In *Proceedings of the 3rd International Conference on Principles of Knowledge Representation and Reasoning (KR'82)*, 165–176.

Richardson, N. E. 2008. *An Operator Induction Tool Supporting Knowledge Engineering in Planning*. Dissertation, University of Huddersfield.

Shahaf, D. 2007. Logical filtering and learning in partially observable worlds. Master's thesis, University of Illinois, Urbana-Champaign.

Sierra. 1995. The incredible machine 3. [CD-ROM].

Vilain, M. B. 1982. A system for reasoning about time. In *Proceedings of the 2nd National Conference on Artificial Intelligence (AAAI'82)*, 197–201.

Wolfe, M. F. 2011. *Rube Goldberg: Inventions*. Simon & Schuster.

Wood, J. 2008. Minimum bounding rectangle. In *Encyclopedia of GIS*. Springer. 660–661.

Wu, K.; Yang, Q.; and Jiang, Y. 2007. ARMS: an automatic knowledge engineering tool for learning action models for AI planning. *Knowledge Engineering Review* 22:135–152.

Zhuo, H. H.; Yang, Q.; Hu, D. H.; and Li, L. 2010. Learning complex action models with quantifiers and logical implications. *Artificial Intelligence* 174:1540–1569.