Characterizing Predicate Arity and Spatial Structure for Inductive Learning of Game Rules

Debidatta Dwibedi and Amitabha Mukerjee

Indian Institute of Technology, Kanpur
debidattadwibedi@gmail.com, amit@iitk.ac.in

Abstract. Where do the predicates in a game ontology come from? We use RGBD vision to learn a) the spatial structure of a board, and b) the number of parameters in a move or transition. These are used to define state-transition predicates for a logical description of each game state. Given a set of videos for a game, we use an improved 3D multi-object tracking to obtain the positions of each piece in games such as 4-peg solitaire or Towers of Hanoi. The spatial positions occupied by pieces over the entire game is clustered, revealing the structure of the board. Each frame is represented as a Semantic Graph with edges encoding spatial relations between pieces. Changes in the graphs between game states reveal the structure of a "move". Knowledge from spatial structure and semantic graphs is mapped to FOL descriptions of the moves and used in an Inductive Logic framework to infer the valid moves and other rules of the game. Discovered predicate structures and induced rules are demonstrated for several games with varying board layouts and move structures.

Keywords: predicate discovery, spatial structure discovery, game rule learning, semantic graphs, multi-object tracking, vision-based ontology discovery, inductive logic programming, kinect

1 Introduction

Any formal system is built on a base vocabulary of predicates, functions and constants. These predicates may show much variability while representing the same linguistic terms. In modeling games with moving pieces, predicates such as `move()` or `adjacent()` may vary in argument patterns and semantics owing to differences between games. Thus, in Tic-tac-toe, a `move` involves adding a piece, whereas in Towers of Hanoi or Kalaha, many pieces may be moved at once. Thus, the arity of `move()` varies across games. Similarly, adjacency relations will change depending on the board layout (1-D, 2-D, mixed-vertical, triangle vs grid, etc.). In order for an ontology to be induced for such games, it is crucial that one start with the right predicates. In addition the range of constant values that a variable can take (e.g. the set of valid positions) has to be specified. In this paper, we look at single-person games involving pieces that move, and we ask if instead of introducing such knowledge implicitly in the background, can we discover such structures by visually observing the game play?

Inductive Logic Programming and allied methods have shown immense advantages in learning domain theories for a wide class of problems [1, 2], but the approach is still restricted by an inability to discover a suitable set of predicates, which require grounding in sensorimotor data. Formal systems with polymorphism permit functions with varying arity, but these cannot be handled efficiently in inductive logic situations. Thus, the background input for inductive logic programming invariably involves predicates with fixed arities.

When a child is shown a game of Tic-tac-toe, that each move involves adding a single piece is immediately obvious, whereas in Towers of Hanoi, it is clear that a move may involve several pieces. Similarly, one glance at a chess board tells a learner that it has 8×8 squares, and that the position of any piece can take a value only from these 64 possibilities. This suggests that some aspects of the vocabulary used in the background theory may be inferred by the learner - as opposed to being programmed - thus providing greater flexibility for inducing the domain theory.

Here, we build on recent work in semantic graph discovery from RGB-D (depth data) images to learn structures of interactions between objects [3, 4] to explore the possibility of learning some aspects of predicate structures in games involving moving pieces. Specifically, we attempt to discover a) the arity and structure of base predicates such as `move()`, and b) the underlying spatial structure that provides the set of constants that define admissible values for some fluents like position. In the process, we also construct visual semantic interpreters and generators for these predicates, in terms of the visual routines which result in a discovered cluster.

The approach is demonstrated in three one-person games (or puzzles) involving spatial reconfiguration of pieces : Jumping frogs (1-D); Towers of Hanoi (1-D with vertical) and 4×4 Peg Solitaire (2-D)(Fig. 1). Both Jumping frogs and Peg solitaire have been modeled in simulation using the BlenSor RGBD simulation system[5]; Towers of Hanoi has been tested both on real and simualted data. The datasets and code used is being made available at `http://www.cse.iitk.ac.in/users/vision/debidatt/`

(a) Jumping frogs puzzle (b) Towers of Hanoi (c) 4×4 Peg Solitaire

Fig. 1: Examples of Spatial Reconfiguration games handled. Board spatial layout and predicate structures such as number of pieces involved in moves are inferred from the visual structure. ILP then is able to infer aspects such as that higher disks must be smaller in Towers of Hanoi.

2 Related Work

Inductive logic programming (ILP) attempts to hypothesize the simplest hypothesis explaining a set of (mostly) positive examples using background knowledge [1, 2]. More formally, given a set of observed examples E_i (propositions), and the categories c_i they belong to, ILP attempts to find the simplest model H (a first-order-logic theory) s.t. for all training pairs $\langle E_i, c_i \rangle, H \wedge Ei \wedge B \models c_i$, while $\forall c' \neq ci, H \wedge Ei \wedge B \not\models c_i$.

ILP approaches have been used in learning the rules for boardgames like Tic-Tac-Toe and Hexapawn [6], dice-based games [7] and card games [8, 9]. In each of these, the backghround knowledge already covers concepts like board representation, adjacency / linearity tests, frame axioms, turns and opponents, piece ownership and spatial predicates. Our objective is to start a bit further back, and try to discover the structure for some these predicates.

However, hypotheses discovered by ILP (Progol) are restricted to essentially single clause hypotheses in the refutation chain, and multi-clause induction is highly inefficient [10, 11]. One approach to multi-clause induction is to prioritize the ordering of rules using a set of meta theoretic rules ("top theory") that enables multi-clause refutations [11]. This has been used in learning grammars and also a strategy for the Nim game. Other attempts to extend the paradigm include interleaving induction with abduction models to generate more compact models for modeling event structures [12]. Systems attempting to learn game strategy are better served by using models related to learning planns, which often use a PDDL structure [13]. However, our objective here is at the vision-logic interface, and not in the domain of logic per se, hence we restrict ourselves to Progol for our testing.

2.1 Inducing domain theories for games from vision

Inducing rules of games using vision as input has been attracting increasing attention in recent years [6, 14, 9], since they provide a key test for other generalizations that may be possible for real-world problems. In Barbu etal [6], the learned rules are used impressively by a robot to manipulate the pieces onto a wooden frame to actually play the game. They use ILP (Progol) to learn valid moves of the game pieces and winning conditions in six games. The approach proposed by Kaiser [14] is also inductive, requiring a few visual demonstrations to learn rules for games such as Connect4 or Gomoku.

However, these systems needs to be provided with the predicate structure implicitly via background knowledge. Thus [6, 15, 14] all assume a 2-D grid of known size, and pre-define the set of possible moves and adjacency relations of interest. The priors embedded in the background knowledge thus restrict the generality of such systems. Also, the visual classifiers associated with each predicate are hard-coded and game specific. We show that as part of ths semantic-graph analysis, these visual routines, (and hence the argument structure) can be discovered for predicates like `move()`.

2.2 Representing Scenes with Semantic Graphs

In a series of recent papers, Aksoy and co-workers [16, 3] have mapped videos to dynamic graphs with nodes representing objects and edges encoding semantic relations such as contact. Related ideas for learning semantic relations by tracking objects can be found in the semantic segmentation of scenes[17], affordance modeling of objects[18] and manipulation planning[19]. Semantic graphs can model manipulation actions[3][4] in terms of primitives like merging and dividing and used to classify higher-order actions like making a sandwich, cutting a cucumber, pouring liquids, etc. When a piece is moved in a game, manipulations are relatively simpler, since the piece does not deform or merge into others.

A key requirement for our work is that objects must be tracked reliably across visual frames. As in [4], we propose to use Kinect-based RGBD image inputs for the tracking. Contact between pieces is important in some games (e.g. Towers of Hanoi), and this is determined by analyzing four types of relationships between each pair: *touching, overlapping, non-touching and absent.* A matrix encoding all possible relation pairs is created and this is compressed to represent only the change in relation pairs. The dynamic changes in graphs caused by manipulation actions are compared by converting these relations into strings. Thus one may define spatial and temporal similarity measures between different actions, and cluster such actions, resulting in a template for game actions such as `move()`. Other candidates for edges in semantic graphs may be obtained by tracking the hand in 3D videos [20].

In the attempt presented here, part of the structure is being learned via the semantic graph in terms of contact and neighbourhood relations, and this is used to identify the type of primitive predicates that would be used to describe the system. These predicates are added to a sparse human-defined ontology of background knowledge in order to learn rules for games and puzzles from the RGBD videos.

We modify the semantic graph for situations specific to rigid piece motions as in games. We are given a set of game videos as input, but are not told about the spatial structure - whether it is being played on a grid or a line or a triangle or other spatial layout. We also do not know the number of pieces involved in each state-transition and their specific behaviours. In the next section, we see how we do this starting with RGBD videos which enable improved 3D tracking since camera-based depth data is available. For example. clustering all the 3D positions of the pieces enable us to obtain the "cells" that a piece can occupy. Grid layouts are identified using Principal Component Analysis; if the layout is aligned to the dominant eigenvectors, it is a grid. Next, we identify if there is direct contact (as in Towers of Hanoi), if so, contact is used as the edge relation in our semantic graph. Else, we use adjacency relations defined on the board discovered. This initial analysis also tells us the number of changes that occur on different types of moves, and how these can be captured in terms of a "move" or a "transition" predicate.

In our work we analyze the RGBD video of a game. If there are contact situations, we consider contact as a primitive for the Semantic graph analysis; else we

use neighbourhoods on the discovered spatial structure. These relationships are mapped to FOL predicates which are then used in an ILP framework to induce rules for the game.

3 Semantic Graphs of game scenes from RGBD video

In order to generate semantic graphs from images or point clouds, the first task is to robustly segment and track each piece. Challenges include occlusion by the hand or by other objects and altered appearance. Other changes come about due to division or merging (e.g. a tower may be a single merged object in Towers of Hanoi). The above problem is simpler in games because pieces are usually rigid. However, many games have pieces that are identical in colour and shape, throwing up other challenges.

3.1 Game Piece Segmentation

With 3D data, object segmentation can be performed to cluster points close to each together based on Euclidean distance[21]. Algorithm 1 is a modified version where we perform filtering based on the colour in the HSV space before the clusters of points are discovered in the scene by doing Euclidean clustering based on distance. This is done because sometimes game pieces of different colours might be placed on top on another or in contact with each other like in the Towers of Hanoi. So our objective is to extract clusters of points as game pieces. These clusters should either have perceptually different colours or be separated above a particular threshold in space as shown in Fig. 2.

Algorithm 1 Pipeline to extract objects from scene

1. Use a Pass Through filter to focus on the table-top.
2. Use RANSAC to filter out points of the table-top from the cloud.
3. Perform Colour-based filtering of the point cloud in HSV space.
4. Do euclidean clustering of the different colour clouds to give objects that are either separated in space or have perceptually different colours.

Fig. 2: Game pieces found in a scene from the real Towers of Hanoi dataset

3.2 Multi-object Tracking

In the multi-object tracking problem, a label associated with an object needs to be linked with the same object in the next frame and this needs to be done with all objects present in the scene. The problem is challenging owing to all pieces being identical in many games, and further complicated due to occlusion by the player's hand or by other pieces. A model-based detection method cannot be used here since many objects have the same shape and colour.

Aksoy et al.[3], extracted segments from the images using super-paramagnetic clustering in a spin-lattice model[22]. Doing this allowed them to perform robust markerless tracking of the segments. A number of other tracking algorithms[4][23] attempt to handle objects that may break up (cutting with a knife) or join together (pouring from one glass to another), etc. Since game pieces are usually rigid our tracker can make the assumption that pieces do not break up or merge significantly.

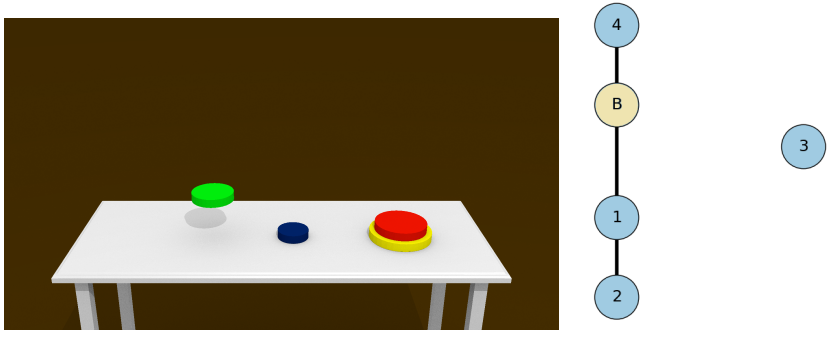
Our proposed method for tracking multiple-objects in a point cloud video is based on the occupancy of voxels by an object in one frame and the next. Multiple object tracking can be reduced to an assignment problem where the objects detected in frame i need to matched with themselves in frame $i+1$.

The assignment problem is a combinatorial optimization problem. It consists of finding a maximum weight or minimum cost matching in a weighted bipartite graph. In other words, there are two sets $A = \{a_1, a_2, .., a_n\}$ and $B = \{b_1, b_2, ..., b_n\}$. There is a certain cost for matching a a_i with a b_j. The assignment problem is to match each members of set A one member of set B such that the total cost of the assignments is minimized. The Hungarian method is used to solve the label assignment problem in polynomial time.

Using Euclidean distance between the centroids [19] may fail if there are multiple objects moving simultaneously. We use the octree overlap between point clouds that is the amount of overlap between axis-oriented bounding boxes of the objects. The hierarchical octree [24] method reduces complexity by downsampling the point cloud. We build the octree representation of the objects found by segmentation in two consecutive frames. If it moves, there is going to be a spatial overlap between the same object in the two consecutive frames. This overlap will be zero with the other objects present in the scene. We use this overlap in space to track objects by maximizing the sum of all overlaps while assigning labels from one frame to the next. There are two assumptions that make this tracking algorithm work. Our objects of interest are non-planar and rigid. Planar objects may have zero overlap with themselves in the next frame. The action performed by the player is slow enough for the Kinect to record the movement of the objects. If the frame-rate of recording the point clouds is slow there will be no overlap. In our case, however, we recorded gameplay at the usual pace a person plays and there was considerable overlap between the same objects in consecutive frames at normal Kinect recording rates. We also suggest the use of a Kalman filter to improve tracking under full occlusion.

3.3 Semantic Graphs

A semantic graph of the scene encodes the relationships between the objects. Building semantic graphs depends on choosing some primitive relations for the edges, and this often depends on the task one is looking at. An intuitive primitive is to consider contact, e.g. Yang et al.[4], but sometimes an object like a bar, may be privileged [19]. In our situation, the table-top is a special object whose contacts are not listed as predicates. Aksoy et al.[3] encode proximity relationships even if they are not in contact. They also encoded the semantic relationship *overlapping* which meant one segment is included in another.



(a) Frame 93 from *Towers of Hanoi* (simulation) dataset

(b) Semantic Graph

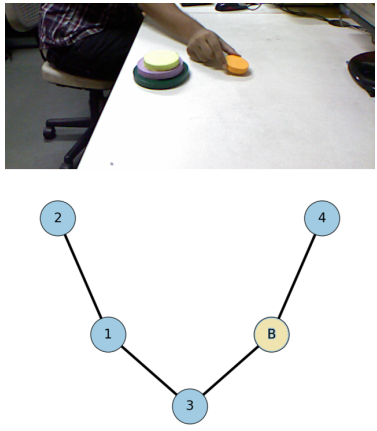
Fig. 3: Example Semantic Graph

In most board games or puzzles the game state is altered by picking up a piece and placing it somewhere else on the board, but sometimes an intermediate piece or the piece at the target square, if of an opposing colour, may be removed. In games such as the Towers of Hanoi, vertical contact occurs frequently, and this needs to be represented.

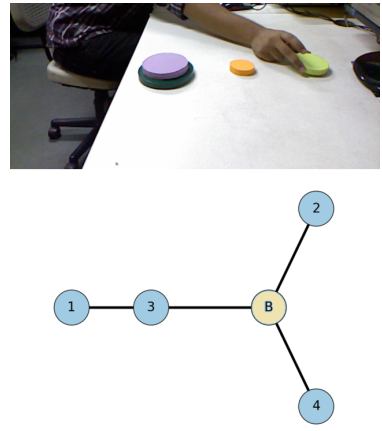
In Fig. 3, there are four pieces from largest piece (1, yellow) to smallest (4, blue) with red (2) and green (3) in between. The board is labeled B. Edges reflect contact between pieces. Thus, the graph shows that a stack of 1,2 is on the board, as well as 4, but the green piece (3) is not in contact with anything. Changes in this semantic graph - e.g. 3 being placed on top of 2 - will represent a move action.

We can now discover the states of the game by looking for configuration changes of the game pieces on the board. Every time a player lifts up a piece, an edge is broken. The moment the player places the piece back on the board or on another piece, a new edge is formed. Hence, game states can easily be discovered from the video by looking for states where the number of edges changes. Each node in the graph also stores meta-information such as the coordinates of its centroid, average colour of the object, number of visible points and the volume occupied by the bounding box of the object in the current frame. After the states

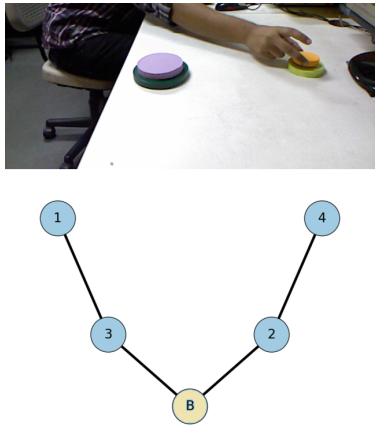
have been detected, the change from one state to another can be found out by looking for changes in the meta-information. In Fig. 4, some game states from the *Towers of Hanoi* dataset, that were discovered automatically, are shown. We



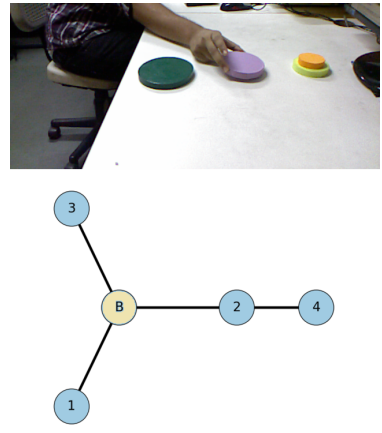
(a) Frame 4 with Semantic Graph



(b) Frame 83 with Semantic Graph



(c) Frame 173 with Semantic Graph



(d) Frame 251 with Semantic Graph

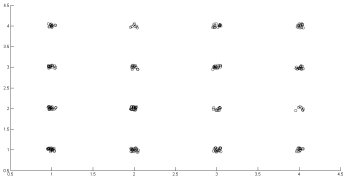
Fig. 4: Automatic detection of game states in the *Towers of Hanoi* Real dataset. Blocks and their labels in the graph:(1,purple),(2,yellow),(3,green),(4,orange). For example, comparing graphs (a) and (b), we find that the move consisted in taking the piece 2 from the stack 3,1,2 to the board.

observe that discovering game states is not a trivial problem. For example in the 4×4 peg solitaire, after a piece has been moved, the intermediate, jumped-over piece is removed. Here the system needs to be told that the intermediate stage does not constitute a "game state". This could also be learned via a heuristic looking at pauses in the game, but as of now, this has not been implemented.

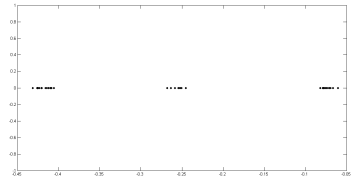
4 Learning Spatial States

Many logical systems start with an implicit assumption about the board on which the game is being played. But this need not be the case. A human observing a game immediately notes the type of board on which the game is being played. Thus, a game such as a 4×4 peg solitaire will have a 2-D structure in the horizontal plane, whereas the Towers of Hanoi has essentially a 1-D structure with vertical contacts. The distribution of spatial locations of the pieces during an entire game can be used to infer the game board, using the following steps:

1. **Discover intrinsic dimensionality of the game**: The system does not have any idea in the beginning whether the game is 1D or 2D or 3D. After it has discovered the game states by using the methods described in the previous section, it populates a list of the positions of all the game pieces across all the game-state frames. These are data points where game pieces have visited during the game play. By performing Singular Value Decomposition(SVD) on these coordinates the intrinsic dimensionality of the game is known. One-dimensional games have only one significant eigenvalue.
2. **Transform from camera coordinates to board coordinates**: X_b, Y_b, Z_b are coordinates of the object in the frame of the board which will be used to find the clusters. These co-ordinates are obtained by transforming the camera coordinates X_c, Y_c, Z_c by using the cosines of the angles between the axes. $\hat{x}_b, \hat{y}_b$ and $\hat{z}_b$ represent the unit vectors of the axes in the frame of the board. $\hat{z}_b$ is obtained as the average of normals of the points on the board. $\hat{x}_b$ and $\hat{y}_b$ are obtained by SVD mentioned above. The eigenvector corresponding to the largest eigenvalue gives $\hat{x}_b$ if it doesn't coincide with $\hat{z}_b$. Similarly, In 2D games the second significant eigenvector gives $\hat{y}_b$. This can also be found as a cross product of $\hat{z}_b$ and $\hat{x}_b$. The above generalizations don't hold true when the game being played doesn't conform to an usual rectangular grid like triangular peg solitaire.
3. **Discover discrete valid positions of game pieces**: The next step is to look for clusters in the positions occupied by game pieces in the game states. While finding out the optimal number of clusters is an open problem, there are statistical methods to estimate the optimum number of clusters in a dataset like ours. One method will be to look for an elbow or a bend in the sum of squared error(SSE) plot. The locations of the clusters are discovered by performing k-means clustering using the value of k found by using the elbow method. In Fig. 5(a) and Fig. 5(b) there are sixteen clusters and three clusters respectively. Fig. 6 shows the elbow method being used to determine the number of clusters in corresponding to the four holes in one dimension in $4 \times 4 PegSolitaire$.
4. **Represent game state**: For each game state, each game piece is assigned to its nearest cluster. Doing so, allows us to generate a general representation of game states of any game. This might leave us with a cluster that is unoccupied which can be represented as empty. We transfer these states to a logic programming system which will be a better domain to induce



(a) 16 clusters in *4×4 Peg Solitaire*



(b) Three clusters in *Towers of Hanoi*

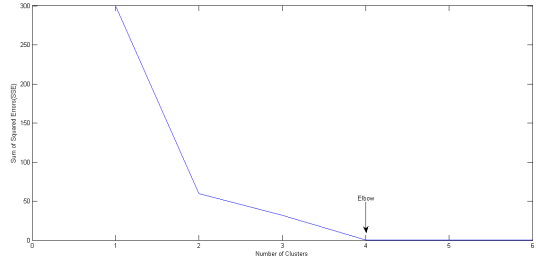


Fig. 6: Elbow method to find number of clusters in one axis of *4×4 Peg Solitaire*

the rules of games. The first game state(Fig. 1(a)) in Four Frogs will be `[{a},{b},{},{c},{d}]` where a, b, c and d are the labels given to the game pieces. The third hole is unoccupied in the beginning which is represented by the empty set. In Towers of Hanoi, the state shown in Fig. 1(b) will be represented by `[{a,b,c},{d},{}]`. This representation is there to handle games where pieces can be placed one on top another occupying the same discrete cluster on the board. This can be extended to 2D games where a matrix of characters will represent the game state.

4.1 From Semantic Graphs to Horn Clauses

We use meta-information contained in the nodes of the graphs and changes in that from one game state to the next to generate logical clauses that will help us learn the rules. We generate the background knowledge and positive examples (instances seen in video) to come up with hypotheses regarding the rules of the game.

The ontology used to represent games and involves three kind of predicates:

1. *Attributes of game pieces* derived from visual classifiers like size, colour, shape, starting position etc.
2. *Relationships between game pieces* generated from the edges of the semantic graphs like *on*, *contact* etc.
3. *Movement of game pieces* generated from changes in game states and semantic graphs (*move*, *transition*, etc.).

Background Knowledge: We assume that game pieces are objects that will need to be monitored. Attributes of the game pieces like color, shape and size may constrain the possible moves it can make. First, we need to identify

the number of pieces. Thus, a 4-piece Towers of Hanoi, may have the following initial declaration: `piece(a). piece(b). piece(c). piece(d).`

In 1-D games, location is described with one variable and in 2-D with two. In the Towers of Hanoi, 3 clusters are discovered on the primary eigenvector. Each cluster is also associated with a number which helps in comparing their position with other clusters. They are declared as follows: `x(l1). x(l2). x(l3). project(l1,1). project(l2,2). project(l3,3).` A set of colours are predefined and associated with a HSV classifier. These are used to declare a colour for each game piece:

```
colour(a,red). colour(b,green). colour(c,yellow). colour(d,blue).
```

Numerical features like size is obtained as the largest dimension of the bounding box of the game piece, rounded off to an integer scale:

```
size(a,1). size(b,3). size(c,9). size(d,10).
```

We do not use shape classifiers in the present analysis since in the games we consider all objects have the same shape. For each numerical feature there is a meta-clause generator that compares their values. For example the clause generated for size is shown below:

```
greatersize(A,B) :- piece(A),piece(B),size(A,NA),size(B,NB),NA>NB.
```

The function *diff* gives us the number of steps a game piece has been moved and in what direction (positive is along the default axis). *absDiff* ignores the direction. In the 4×4 peg-solitaire *diff* and *absDiff* operate on each dimension separately. In the towers of hanoi we also use predicates for *top* and *bottom* in a stack.

```
diff(X1,X2,Diff):- x(X1),x(X2),project(X1,N1),project(X2,N2),
Diff is N1-N2.
abs(X,X) :- X>=0.abs(X,Y) :- X<0, Y is -X.
absDiff(X1,X2,Diff) :- x(X1),x(X2),project(X1,N1),project(X2,N2),
Diff1 is X1-X2, abs(Diff1,Diff).
neighbour(X1,X2) :- absDiff(X1,X2,1).
top(A,[A]).top(A,[B|C]) :- top(A,C).
bottom(A,[A]).bottom(A,[B|C]) :- bottom(A,B).
```

Note that for 2D games, the *diff* is modified *xDiff* and *yDiff* and similarly for *absDiff*.

Given a set of observations we can obtain **Positive examples** of board play. A critical inference has to do with valid **Moves of game pieces**. A move results in a transition from one spatial graph to another, which includes a piece move along with possible side effects (e.g. removal of the intermediate piece in 4×4 peg solitaire). The relationship *transition* encodes the active piece and the states of clusters that undergo change from one game state to the next. It has the following structure:

```
transition(<active pieces>,<initial states>,<final states)>.
```

The predicate shown below is from the Towers of Hanoi game and represents a piece d being moved where the set of game pieces at the initial position $l1$ was `[a,b,c,d]` and that at final position $l2$ after the move was `[d]`:

```
transition(d,[a,b,c,d],[],[a,b,c],[d]).
```

The arity of the transition predicate varies from game to game. In the 4×4 Peg

Solitaire, the number of pieces involved in a move are two and the number of positions where there is change from one game state to the next is three. Hence, the *transition* relation example for the move where piece $p1$ in position $l1$ jumps over piece $p2$ in $l2$ to land in $l3$ following which $p2$ is removed looks like this:
`transition(p1,p2,[p1],[p2],[],[],[],[p1]).`

Table 1: Games learnt with their respective modes of data generation

Game	Nature of Dataset
Towers of Hanoi	Animated(generated in Blensor), Real(recorded with a Kinect)
Four Frogs	Animated(generated in Blensor)
4 × 4 Peg Solitaire	Game traces of a simulation

5 Experiments and Results

5.1 Towers of Hanoi

In addition to one real game played, we used the RGBD simulator BlenSor[5] to animate four differently sized blocks with *Towers of Hanoi* puzzle being solved. There are 740 frames of 640 × 480 RGBD images recorded on an artificial Kinect sensor in BlenSor. The real Kinect data with the Towers of Hanoi being by a person has 1200 frames. The spatial structure discovery has been shown earlier. The ILP system input includes the following:

```
colour(a,yellow).colour(b,red).colour(c,green).colour(d,blue).
size(a,10).size(b,8).size(c,4).size(d,2).
on(d,a).on(d,b).on(d,c).on(c,b).on(c,a).on(b,a).
from(d,[a,b,c,d],[d]).from(c,[a,b,c],[c]).from(d,[d],[c,d]).
```

The rules learnt by PROGOL are:

```
on(A,B) :- greatersize(B,A).
transition(A,B,C,D,E) :- top(A,C), top(A,E).
```

The first rule translates as "No disk may be placed on top of a smaller disk." The second rule says that piece A moves from the top of the stack C and to the top of stack E.

5.2 Jumping Frogs puzzle

The animated dataset consists of 560 frames of 640 × 480 RGBD images. There are five cylindrical holes in a row, two red pegs (which can only move right) and two blue pegs (only move left)(Fig. 1(a)). Initially, the red pegs are placed in the two left holes and the blue pegs are placed in the two right holes leaving a hole in between that is empty. The goal of the game is to swap the positions of the red pegs with the blue pegs. PROGOL generalizes the clause *move* and comes up with four rules:

```
move(A,B,C) :- diff(B,C,-2), colour(A,blue).
move(A,B,C) :- diff(B,C,-1), colour(A,blue).
```

```
move(A,B,C) :- diff(B,C,1), colour(A,red).
move(A,B,C) :- diff(B,C,2), colour(A,red).
```

We learn that if there is an object that moves right its colour must be red and if there is one which moves left then its colour must be blue. More interestingly, the system discovers that there are two types of moves a piece is able to do that is one step and one jump which implies moving two steps at the same time.

The colours of the pegs were then interchanged. The rules learnt by appending the newer clauses with the older ones are:

```
move(A,B,C) :- diff(B,C,-2), startpos(A,l1).
move(A,B,C) :- diff(B,C,-2), startpos(A,l2).
move(A,B,C) :- diff(B,C,-1), startpos(A,l1).
move(A,B,C) :- diff(B,C,-1), startpos(A,l2).
move(A,B,C) :- diff(B,C,1), startpos(A,l4).
move(A,B,C) :- diff(B,C,1), startpos(A,l5).
move(A,B,C) :- diff(B,C,2), startpos(A,l4).
move(A,B,C) :- diff(B,C,2), startpos(A,l5).
```

Thus the colour dependence is replaced by a clause for the row where the pieces start from. This highlights the fact how the rules learnt by induction learning can undergo radical changes depending on the dataset

5.3 4×4 Peg Solitaire

In the beginning, of this game there are 15 marbles arranged in form of a 4×4 grid with one position empty(Fig. 1(c). The marbles can only move by jumping to an empty position and by doing so the piece over which they jumped is removed. The objective is to remove as many pieces as one can, preferably reaching a single piece. We use game traces of a simulation of this game being solved to test how good our system is in inducing the rules in case it has perfect information regarding the game states. The rules learnt by ILP are:

```
move(A,B,C):- xabsdiff(B,C,2). move(A,B,C):- yabsdiff(B,C,2).
transition(A,B,C,D,E,E,E,C):-piece(A),piece(B),top(A,C),
            bottom(A,C),top(B,D),bottom(B,D),empty(E).
```

The two move rules have learned that the moves take place either horizontal or vertical rows of three neighbouring cells. In the transition predicate, the arguments are the pieces involved (here A,B), and the remaining 3+3 arguments are the pieces at the three locations involved, before and after the move. Thus the learned rule says that the state of loc1 and loc2 changes to E, which was the initial state of loc3. The state E is identified as the special constant `empty()` at the end of the rule. The piece at loc3 becomes C which was initially at loc1 (i.e. the piece A is moved to loc3). The top and bottom rules are used to assert equivalence - basically A and C are colocated, as are B,D. Thus, the rule infers that A moves from loc1 to loc 3, and that the piece B is removed from the jumped-over position loc2. The three locations are arranged in a horizontal or vertical row of the board.

5.4 Discussion

We observe that in all three cases, the spatial structure can be inferred at the visual level, permitting a set of constants which the position attributes in move() etc can be assigned to. Also the number of pieces and positions affected by move are identified in the vision system. When the resulting game states and transitions are introduced into the ILP system, we find that it is able to derive the right rules, such as identifying that in ToH, the higher disks must be cmaller, or that in peg solitaire, adjacency relations (for move) are only row or column-wise. Similarly, in the peg solitaire, the fact that the jumped-over piece (also an argument to move) is removed, is inferred.

6 Conclusion

One of the major challenges in inducing knowledge representations involves discovering the right set of logical primitives to be used. Here we have presented a framework that is able to analyze RGBD videos of game scenes using dynamic semantic graphs, which permit generation of suitable Horn Clause structures. The system uses an improved tracking based on the assumption that game pieces do not change shape or visual attributes (like colour or shape). We then demonstrate its application in learning the rules of game and puzzles. The system can successfully induce the spatial description of boards for 1-D and 2-D games, and also induce vertical contact situations and their ramifications for an otherwise 1-D game such as Towers of Hanoi. The arity of predicates such as "move" varies in these games and is captured via the pre-processing in the Semantic Graph step.

As of now, we have demonstrated this for only three simple games. A number of loose ends remain in the present implementation. As of now, the end states of a game are not being discovered, hence we are not able to generate a Game Description Language(GDL) which will enable the system to start playing these games. In most real situations, the learner often needs to be told about the start and end configurations along with whether it was a winning or losing game, etc. Our system can be enhanced with this start and goal state knowledge to generate the suitable GDL for automatic game playing. Further, the system cannot handle multi-player games, which require event calculus representations. However, our main focus has been to demonstrate the idea of obtaining descriptors with the correct number of arguments, which would apply equally to event calculus or other planning formalisms.

Also, for any system using vision, improvements are always possible in tracking. Recent research[23][25] on multi-object tracking has shown encouraging results which may be helpful in tracking for games with more game pieces.

However, the main contribution of this work is at the level of the implicit knowledge used in defining logical descriptors. This is a challenging problem for knowledege representation in general that has not been adequately investigated, and this work takes some initial steps in developing vision-based approaches towards discovering this implicit structure.

References

1. De Raedt, L.: Inductive logic programming. In: Encyclopedia of machine learning. Springer (2010) 529–537
2. Muggleton, S.: Inverse entailment and progol. New generation computing **13**(3-4) (1995) 245–286
3. Aksoy, E.E., Abramov, A., Dörr, J., Ning, K., Dellen, B., Wörgötter, F.: Learning the semantics of object–action relations by observation. The International Journal of Robotics Research **30**(10) (2011) 1229–1249
4. Yang, Y., Fermuller, C., Aloimonos, Y.: Detection of manipulation action consequences (mac). In: CVPR 2013. (2013)
5. Gschwandtner, M., Kwitt, R., Uhl, A., Pree, W.: BlenSor: Blender Sensor Simulation Toolbox Advances in Visual Computing. Volume 6939 of Lecture Notes in Computer Science. Springer Berlin / Heidelberg, Berlin, Heidelberg (2011) 199–208
6. Barbu, A., Narayanaswamy, S., Siskind, J.M.: Learning physically-instantiated game play through visual observation. In: Robotics and Automation (ICRA), 2010 IEEE International Conference on, IEEE (2010) 1879–1886
7. Santos, P., Colton, S., Magee, D.: Predictive and descriptive approaches to learning game rules from vision data. In: Advances in Artificial Intelligence-IBERAMIA-SBIA 2006. Springer (2006) 349–359
8. Magee, D., Needham, C., Santos, P., Cohn, A., Hogg, D.: Autonomous learning for a cognitive agent using continuous models and inductive logic programming from audio-visual input. In: Proceedings of the AAAI workshop on Anchoring Symbols to Sensor Data. (2004) 17–24
9. Hazarika, S.M., Bhowmick, A.: Learning rules of a card game from video. Artificial Intelligence Review **38**(1) (2012) 55–65
10. Yamamoto, Y.: Research on Logic and Computation in Hypothesis Finding. PhD thesis
11. Muggleton, S.H., Lin, D., Tamaddoni-Nezhad, A.: Mc-toplog: Complete multi-clause learning guided by a top theory. In: Inductive Logic Programming. Springer (2012) 238–254
12. Dubba, K., Bhatt, M., Dylla, F., Hogg, D.C., Cohn, A.G.: Interleaved inductive-abductive reasoning for learning complex event models. In: Inductive Logic Programming. Springer (2012) 113–129
13. Edelkamp, S., Kissmann, P.: Symbolic exploration for general game playing in pddl. In: ICAPS-Workshop on Planning in Games. Volume 141. (2007) 144
14. Kaiser, Ł.: Learning games from videos guided by descriptive complexity. In: Twenty-Sixth AAAI Conference on Artificial Intelligence. (2012)
15. Björnsson, Y.: Learning rules of simplified boardgames by observing. In: ECAI. (2012) 175–180
16. Aein, M.J., Aksoy, E.E., Tamosiunaite, M., Papon, J., Ude, A., Worgotter, F.: Toward a library of manipulation actions based on semantic object-action relations. In: IROS-2013. (2013) 4555–4562
17. Delaitre, V., Fouhey, D.F., Laptev, I., Sivic, J., Gupta, A., Efros, A.A.: Scene semantics from long-term observation of people. Computer Vision–ECCV 2012 (2012) 284–298
18. Koppula, H.S., Gupta, R., Saxena, A.: Learning human activities and object affordances from rgb-d videos. The International Journal of Robotics Research **32**(8) (2013) 951–970

19. Dantam, N., Essa, I., Stilman, M.: Linguistic transfer of human assembly tasks to robots. In: Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on, IEEE (2012) 237–242
20. Oikonomidis, I., Kyriazis, N., Argyros, A.A.: Efficient model-based 3d tracking of hand articulations using kinect. In: BMVC. (2011) 1–11
21. Rusu, R.B.: Semantic 3D Object Maps for Everyday Manipulation in Human Living Environments. PhD thesis, Computer Science department, Technische Universitaet Muenchen, Germany (October 2009)
22. Dellen, B., Erdal Aksoy, E., Wörgötter, F.: Segment tracking via a spatiotemporal linking process including feedback stabilization in an nd lattice model. Sensors **9**(11) (2009) 9355–9379
23. Koo, S., Lee, D., Kwon, D.S.: Multiple object tracking using an rgb-d camera by hierarchical spatiotemporal data association. In: Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on, IEEE (2013) 1113–1118
24. Meagher, D.: Geometric modeling using octree encoding. Computer graphics and image processing **19**(2) (1982) 129–147
25. Papon, J., Kulvicius, T., Aksoy, E.E., Worgotter, F.: Point cloud video object segmentation using a persistent supervoxel world-model. In: Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on, IEEE (2013) 3712–3718