Using Genetic Programming to Learn Models Containing Temporal Relations from Spatio-Temporal Data

Andrew Bennett and **Derek Magee** [1]

Abstract. In this paper we describe a novel technique for learning predictive models from non-deterministic spatio-temporal data. Our technique learns a set of sub-models that model different, typically independent, aspects of the data. By using temporal relations, and implicit feature selection, based on the use of 1st order logic expressions, we make the sub-models general, and robust to irrelevant variations in the data. We use Allen's intervals [1], plus a set of four novel temporal state relations, which relate temporal intervals to the current time. These are added to the system as background knowledge in the form of functions. To combine the sub-models into a single model a context chooser is used. This probabilistically picks the most appropriate set of sub-models to predict in a certain context, and allows the system to predict in non-deterministic situations. The models are learnt using an evolutionary technique called Genetic Programming. The method has been applied to learning the rules of snap, and uno by observation; and predicting a person's course through a network of CCTV cameras.

1 Introduction

Learning predictive models from spatial-temporal data is, in general, a hard problem. Events and activities can have variations in their spatial, and temporal scope; include multiple (variable numbers of) objects; can overlap temporally with other events, and activities; and happen in a non-deterministic manner. A model for predicting spatio-temporal events must support this complexity. Our novel technique learns a set of sub-models that model different, typically independent, aspects of data. The sub-models can, in addition to object properties, use temporal relations to describe the scene, and implicit feature selection, based on the use of 1st order logic expressions, to make them robust to irrelevant variations in the data. To combine the sub-models into a single model a context chooser is used. This picks the most appropriate set of sub-models to predict in a certain context, and allows the system to predict in non-deterministic situations. Using the combination of sub-models and the context chooser also reduces the complexity of the model search space, and allows the system to learn a global sub-model that matches most of the dataset, and then learn simple sub-models to cover the cases where the global sub-model does not work.

This approach extends our previous work [2], by allowing a qualitative, as well as a markovian representation of time. This is done by replacing the step-wise markovian view with temporal relations like Allen's intervals [1], and a set of four additional relations to relate the temporal state of objects to the current time. We use Genetic Programming to learn the models, and present an improved fitness function. The system has been successfully tested on handcrafted snap, and uno datasets, along with learning from video the structure of a set of mock CCTV cameras.

There has been much previous work on learning from spatio-temporal domains. Traditional methods usually require a fixed dimensionality vector, existing with canonical ordering / constant meaning, to represent the world. To construct this vector often requires knowledge of the domain, making these methods hard to use in a problem domain where the structure of the domain is variable, and not known a priori. One approach to modelling data of variable dimensionality is to take statistics of a variable size set [8]. This produces a fixed set description, however spatial relationship information is lost in this process. If this information is important within a domain this leads to a poor model. Feature selection can be used to find the most relevant subset of the data, which then allows for a more general model to be built. However, the relevant subset may change from one context to another.

Temporal modelling approaches such as Markov chains, Hidden Markov Models (HMMs) and Variable Length Markov Models (VLMMs) [7] use a description based on graphs to model state transitions. These methods also usually need a fixed dimensionality vector with canonical ordering for each observation. There does not have to be a fixed dimensionality for every observation vector, as theoretically each observation vector can have a different number of dimensions. It is possible to optimise their structure by using local optimisation approaches based on information theory [3]. In VLMMs this optimisation acts as kind of temporal feature selection, but as the input variables stay in the same fixed order spatial feature selection is not performed.

Bayesian networks are a generalisation of probabilistic graph based reasoning methods like HMMs and VLMMs. Again these networks require a fixed input vector, but again their relational structure can be optimised by local search [12], genetic algorithms [5], or MCMC [6] usually based on information theoretic criteria.

An alternative to using graph based methods is to use (1st order) logical expressions. Feature selection is implicit in the formalism of these expressions. Logical expressions also make no assumptions about the ordering of variables, so there is no need to have a have them in a fixed ordering. Progol [14] and HR [4] are Inductive Logic Programming (ILP) methods. In general ILP takes data and generates a set of logical expressions describing the structure of the data. Progol does this by iterative subsumption using a deterministic search with the goal of data compression. HR does this by using a stochastic search using a number of specialist operators. This is similar to Genetic Programming which is described below. These approaches suffer from a number of disadvantages. Firstly, logical expressions are deterministic, so it is hard for then to model non-deterministic situations. However, there has been much work on combining (1st order)

[1] University of Leeds, UK, email: {andrewb,drm}@comp.leeds.ac.uk

logic and probability to solve this problem [16] and [9]. Secondly Progol's search is depth bounded, which limits the size of problems it can work on, as explained in [15]. Thirdly Progol's fitness function is only based on how well the model compresses the data, and not how well the model predicts the data. This can cause incorrect, or invalid models to be produced.

Genetic Programming (GP) [10] is a evolutionary method, similar to genetic algorithms, for creating a program that model a dataset. In a similar way to HR, it takes a dataset data, a set of terminals, and a set of functions; and using a set of operators generates a binary tree that models the data.

Qualitative representations can be used to describe spatio-temporal data in an abstract manner. [1] describes a set of seven temporal relations to represent temporal interactions between objects.

There has been previous work in learning of spatio-temporal models from video by [15] who produced a system that could learn basic card games. It had three parts: an attention mechanism, unsupervised low-level learning, and high-level protocol learning. The attention mechanism uses a generic blob tracker, that locates the position of the moving objects. From this a set of features including: colour, position and texture are extracted. The data is clustered into groups. Using these clusters new input data is assigned its closest cluster prototype. A symbolic data stream is then created by combining together the clustered data, with time information. The symbolic stream is passed to Progol, which builds a model of the data. Once the model has been learnt it can be applied to new data. This allows the system to interact in the world.

[17] looked at learning event definitions from video. A raw video of a scene is converted into a polygon representation. This is then transformed into a force-dynamic model which shows how the objects in the scene are in contact with one another. Using this data and-meets-and (AMA) logic formulae describing the events are learnt using a specific-to-general ILP approach. Work in the area of learning from spatial-temporal data, such as the previous two approaches have inspired our work.

The reminder of this paper will take the following form. The second section looks at previous work about the architecture for the models. The subsequent section looks at an extension to this work to incorporate temporal relations into the sub-models. The subsequent section describes how these models are learnt by Genetic Programming. The subsequent section presents an evaluation of our system, and the final section shows the conclusions of the work and the further work.

2 Architecture for Models of Spatio-Temporal Data

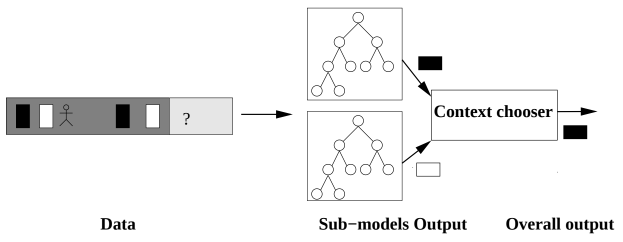

Figure 1. This figure shows the architecture of our model. It has two parts: a set of sub-models, and a context chooser to decide how to use the sub-models in different situations.

An architecture to represent a model of spatio-temporal data, along with associated learning methods is described in our previous work [2]. We use this architecture as shown in Figure 1. It is broken down into two parts: the sub-models, and the context chooser. The sub-models each model a separate part of the underlying process generating the data. Each sub-model contains two sections: a search section, and an output section. The search section looks for a particular pattern in the dataset. A query language, created by ourselves, having some similarity to SQL and Prolog, is used to describe the actual search, and a binary tree is used to represent it. The output section describes what is implied if the search returns true. This will be a set of entities and relations, and their properties the sub-model predicts. Figure 2 shows an example of a sub-model.

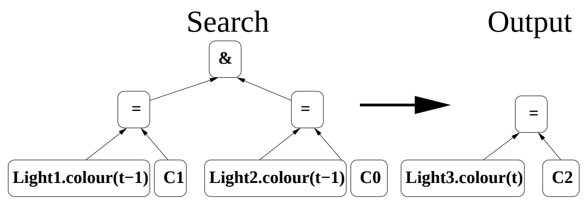

Figure 2. This shows a sub-model matching a traffic light with colour c1, and a different light having a colour c0 both at current time - 1. If the expression evaluates true it will output a new light which has a colour c2, at the current time.

The context chooser is used to decide how to combine the sub-models in different situations. It takes as its input a boolean vector describing which sub-models have evaluated true, and returned outputs, and using a probability distribution decides which ones will form the overall output. A context S_n is defined as a set of sub-models M producing an output in a given context, for example $S_n = M_1, M_2$ represents that M_1, and M_2 have search sections that have evaluated true at the same time. For each context a probability distribution over the possible combinations of model outputs for that context is defined, for example $P_n(M_1), P_n(M_2), P_n(M_1, M_2)$, where $\sum_j P_n(j) = 1$. This distribution is formed from the frequency of occurrence of each situation in the training data in the given context. This can be implemented as a sparse hash table.

3 Incorporating Temporal Relations into Sub-models

To evaluate the sub-models history data from the world is required. The search section of the sub-model uses data pointers to reference particular data items in the history. The search section of the sub-model is then evaluated with respect to this data. If the search section evaluates true, then the output section is implied. In our previous work [2] each data pointer could only reference fixed quantified time points in the history, as shown in Figure 2. The use of this qualitative markovian representation of time implies an exact ordering of the events. When multiple independent events are happening simultaneously this representation will fail, and an alternative method of representing temporal ordering is necessary. In order to quantify temporal ordering in the data we use a combination of Allen's intervals [1], and four novel temporal state relations. Allen's intervals describe temporal relations between objects. There are seven relations which are: meets, starts, finishes, during, before, overlaps, and equal to. Along with describing temporal relations between objects in the history, we need to describe how the objects relate to the current time. An object goes through a series of temporal states, based on how its start and end time relates to current time, these are described Figure 3. Firstly the object is entering the world, its end time is unknown, but its start time is the same as the current time. Secondly the object exists in the world, again the end time is unknown, but its start time is less than the current time. Thirdly the object is leaving the world and its end time is equal to the current time. Finally the object has left the world, where both its start, and end times are less than the current time.

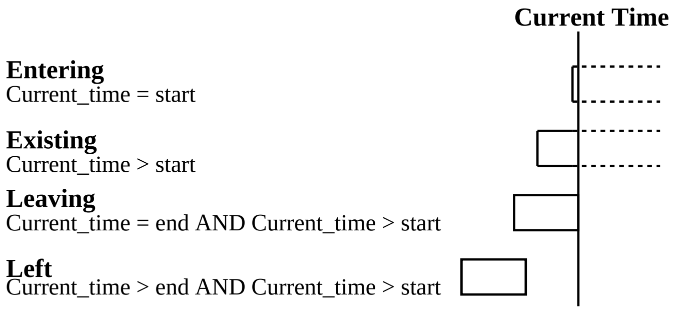
Current Time
Entering
Current_time = start
Existing
Current_time > start
Leaving
Current_time = end AND Current_time > start
Left
Current_time > end AND Current_time > start

Figure 3. This shows the four temporal states, with respect to current time, an object can be in: entering, existing, leaving, and left. The dotted lines represent that we don't know when the object will leave the world.

Both the Allen's intervals, and our additional temporal state relations, are represented in the system as functions of the data, that appear in the search section of the sub-models. These relations do not appear in the data; only the temporal range of individual objects occurs in the data. As the data pointers can be used over the entire history, it is quite likely that a sub-model will evaluate on many different parts of the history. To resolve this issue we just use the result which includes the most recent data. The justification for this is the sub-model will have already output this information at a previous time in other situations.

4 Learning the Models from Data

Previously in our previous work [2] it has been shown that it was intractable to find the set of optimal sub-models by exhaustive search, for all but the simplest problems. The search space is complex, so a stochastic search method was chosen as an alternative. We use Genetic Programming [10], which has already been successfully used for pattern recognition tasks [11].

Genetic Programming (GP) [10] evolves a population of programs until a program with the desired behaviour is found. It is a type of genetic algorithm, but the programs are stored as binary trees, and not as fixed length strings. Functions are used for the nodes, and terminals (for example constants, and variables) are used for the leaf nodes. In order for the population to evolve a fitness function (in our case a predictive accuracy score) must be defined. This score will be used by the GP system to decide which programs in the current generation to use to produce the next generation, and which ones to throw away. To initialise the system, a set of randomly generated programs must be created. Each then receive a score using the fitness function. Algorithms including crossover, mutation and reproduction use the programs from the current generation to create a new generation. Crossover takes two programs and randomly picks a sub-tree on each program, these two trees are swapped over, creating two new programs. Mutation takes one program, randomly picks a sub-tree on it, and replaces it with a randomly generated sub-tree. Reproduction copies a program exactly as it is into the new generation. The programs in the new generation are then scored based on how well data is predicted, and the process is repeated. The GP system will stop when a certain fitness score is reached, or a certain number of generations has passed.

In our implementation of GP we assume that a program is a model containing a context chooser, and a set of sub-models. To initialise the population we generate a set of models just containing one randomly generated sub-model. The sub-model is produced using Koza's ramped half and half method [10]. We apply a hierarchical structure to our sub-models in a similar manner to [13], to try and cut down the search space, and to make finding a solution more efficient.

A set of operators is then used to evolve the population. There are two kinds of operators. Firstly there are operators that try to optimise sub-models which are used in the model, and secondly there are operators that optimise the sub-models themselves. A technique called tournament selection [10] is used to pick a model from the population. Tournament selection picks n models at random from the population, and returns the one with the lowest score, for our experiments we set n to be 5. The operators used to optimise sub-models which are used in the model are shown below:

Reproduction A set number of models are picked via tournament selection and copied directly into the new population.

Adding in a sub-model from another model Two models are picked by tournament selection. A sub-model from the first picked model is randomly selected, and added to the second chosen model.

Replacing a sub-model Again two models are picked by tournament selection, and a sub-model from the first chosen model is then replaced by a sub-model randomly selected from the second chosen model.

Removing a sub-model A sub-model is picked by tournament selection, and a randomly selected sub-model is removed.

The only operator used to optimise the sub-models themselves is crossover. In crossover two models are picked using tournament selection. A sub-model from each model is then randomly selected, and standard crossover [10] is performed on these sub-models.

To score the models a fixed length window is randomly moved over the dataset. At each generation two random locations are picked: one for training, and one for testing. In the training phase the probability distribution used in the context chooser is calculated. In the testing phase the fitness of a model (m) is evaluated over a windowed section of the dataset (w). For each position in the window the model is given a set of history data (h), calculated from the window, and is queried to produce a prediction. This produces a set of possible corresponding outputs (o), and a set of possible corresponding output likelihoods (ol). The similarity (C) of each output with the actual output (r), is computed using the $FindBestMatch$ function, as shown in Equation 1. This function takes the set of actual output, and the set of model output, and firstly pads out them out with blank data so that they are the same size. Then for each item in the actual output set, a unique match in the model output set is found. For each of the matches a comparison is done between the two objects. The comparison looks at how similar each of the properties in the two objects are. Each of the comparisons are summed together to produce a score that shows how good that set of matches is. An exhaustive search is then performed over all the possible combination of matches to find the best (maximal) matching score. The result is then multiplied by its output likelihood. From this the best (maximal) output is found. This is then repeated over the rest of window, and the results summed and then normalised to produce (S), as shown in Equation 2. This fitness function is an improved version to the one described in our previous work [2], as it can be applied to non-deterministic datasets.

$$C(o,r) = FindBestMatch(o,r) \tag{1}$$

$$S(m,w) = \frac{1}{|w|} * \sum_i Max_n(ol_n * C(o_n, r_i)) \tag{2}$$

The system runs in two stages, and will stop running once it exceeds a maximum number of generations. Firstly the system is initialised in the manner described above, and then for five generations it works out the best set of sub-models to use in the models. To do this the system uses reproduction (10%), removing (10%), adding (40%), and replacement (40%). Next the system will optimise these models to find the best solution. It uses crossover (60%), reproduction (10%), removing (10%), adding (10%), and replacement (10%).

5 Evaluation

Our method was evaluated on three different datasets, which were: handcrafted uno data, handcrafted snap data, and data from people walking through a network of mock CCTV cameras. More detail about these datasets is presented in the following section.

5.1 CCTV Data of a Path

A 10 minute video of people walking along a path containing a junction was filmed. This was then used to mock up a network of CCTV cameras. Figure 4 shows a frame from the video. Virtual motion detectors, representing CCTV cameras, were hand placed over the video has shown in Figure 4. Using frame differencing, and morphological operations, the video was processed to determine the location of the motion. If the number of moved pixels in a region exceeded a fixed threshold then the virtual detector outputted that motion had occurred at that location. Hysteresis on the motion detection is implemented as a 2 state, state machine (where the states are motion/no motion). The state machine requires a numbers of frames (normally 10) of stability to change state. The data produced is then placed in a datafile with a motion event recorded per state change going from no motion to motion. This was used to create a training datafile containing 84 state changes and a test file containing 46 state changes.

5.2 Snap

The snap dataset was handcrafted, but the format of it was similar to the snap dataset used in the work of [15]. The snap sequence is the following: initially the computer will see a blank scene, then it will hear the word play, next two coloured cards will be seen. Either they will be both put down at the same time, or put down one by one. If they are the same then the word “equals” will be heard, otherwise “different” will be heard. Then the cards are removed, again either one by one, or at the same time. We ask the computer to only learn the sections where a human is speaking, as it would be impossible to accurately predict the next two cards because they are essentially random. Again three datasets were prepared: a non-noisy, and noisy training set, and a non-noisy test set. All the datasets contained around 50 rounds of snap. The noisy data was generated by adding 10% noise to the non-noisy training set. The noise took the form of removing cards, removing the play state, and changing the output state, for example making the output not equal when it should be equal.

5.3 Uno

The handcrafted uno dataset has a similar sequence to the snap dataset. Again the computer will initially see a blank scene. Then play will be heard. Next two cards, each one having one of three possible coloured shapes on them, will be placed down either at the same time, or one by one. If the two card have the same coloured shape on them the “same” is heard; or if they have shapes of the same colour then “colour” is heard; or if they have the same shapes on then “shape” is heard; or if the cards are different then “nothing” is heard. The cards are then removed either together, or one by one. Three datasets were created: a non-noisy training set, a noisy training set, and a non-noisy test set. Each one contained around 50 rounds of uno. Again noisy data was prepared by adding 10% of noisy data to the non-noisy training data. The noise took the same form as the noisy snap data.

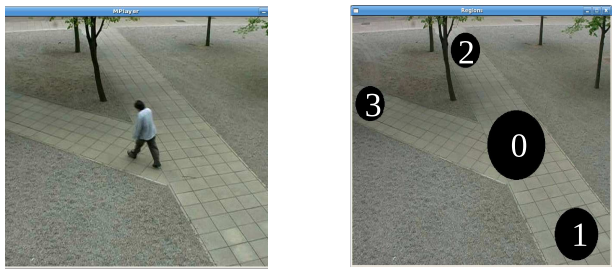

Figure 4. This figure firstly shows a frame of the video with a person taking a decision at the junction point, and secondly it shows where the virtual detectors are on the video.

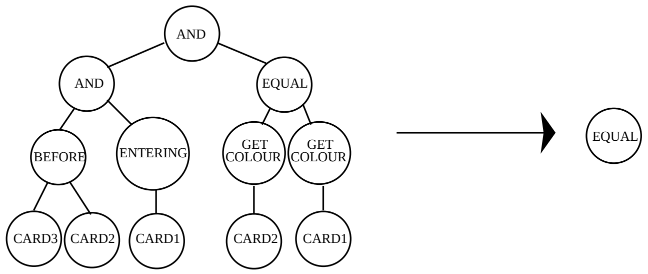

Figure 5. This shows one of the sub-model results for the snap dataset. It is predicting the equal state, by using the properties of three cards. If card3 occurs before card2; and card1 has just entered the world; and the colour of both card1, and card2 is the same then the sub-model evaluates true, and Equal will be returned.

6 Results

To test the system five runs were allocated to each possible combination of dataset. For each run a different random number seed was used to initialise the system. The tests were run on a 2GHz machine having 8GB memory.

To evaluate how well the models have been learnt they were tested on a separate test set. Two metrics were used to evaluate the results: coverage, and prediction accuracy. Coverage (C) scores if the system can correctly predict the dataset (ie. the probability of correct prediction is greater than 0%) and is the number of correct predictions (pc) divided by the dataset size (d) as shown in Equation 3. Prediction accuracy (A) scores with what probability the correct prediction is made, and is the sum of the likelihoods of each correct prediction (pl) divided by the dataset size, as shown in Equation 4. In non-deterministic scenarios this will not be 100%.

$$C = \frac{pc}{d} \tag{3}$$

$$A = \frac{pl}{d} \tag{4}$$

Both the snap datasets were tested on a population size of 4000, and the system was run for 65 generations, taking around 5 hours to do each run. All the runs using the non-noisy datasets were successful. However the models did not get 100% coverage because they failed to produce any output at the start of the test dataset as there was insufficient items in the history. Figure 5 shows an example of this, as it will only evaluate once there are three cards in the history. Four of the results did not predict the first two items in the test dataset, and one of the results only failed to predict the first item. Two out of the five runs using the noisy snap dataset got an exact solution. The noise effected the models causing the sub-models to model incorrect parts of the dataset. This was because some of the noise added to the noisy training set changed the outcomes for some rounds of snap, this then causes the system to model this noise, and to incorrectly predict the outcomes in the test set. Again, like in the non-noisy snap models there was problems predicting the start of the test dataset. The models themselves made use of both the Allen's intervals, and the temporal state relations. Figure 5 shows one of the sub-models produced from the non-noisy snap training set. It shows the use of Allens intervals (the before relation), and the temporal state relations (the enter relation). Most of the models contained four sub-models in them.

The uno datasets were run on a population of size 6000, and for 65 generations, taking around 7 hours to do each run. One out of five runs on the non-noisy dataset managed to get the correct solution, but it did not get 100% coverage because it did not have enough history at the start of the test set to predict the initial items. The rest of the non-noisy results were very close to the solution, and probably needed more generations to find the exact solution. The models themselves were very similar to the models produced for the snap datasets. Both Allen's intervals, and the temporal state relations were used. None of the runs for the noisy dataset managed to produce an exact result, with the noise causing the sub-models to model incorrect parts of the dataset.

The runs using the path dataset used a population size of 2000, and the system was run for 65 generations, taking around 3 hours to do each run. All the runs using the non-noisy dataset predicted well in the main section of the test dataset, but failed to predict well at the start of the test dataset, due to lack of history. Some of the runs also failed to predict infrequently occurring actions in the test set. In the runs using the noisy training set all the models learnt the frequently occurring actions, but they all started to learn some of the noise in the dataset, and this effected their scores on the test dataset. Both the non-noisy and noisy models used Allen's intervals, and the temporal state relations.

7 Conclusions

We have extended the previous work of [2] and shown that that it is possible, by the use of temporal relations, to use a qualitative, as well as a markovian representation of time. This technique is important for a number of reasons. Firstly it produces models that are robust to irrelevant variations in data. Secondly, it allows the system to learn from a dataset containing single actions, and then be able to predict from a dataset containing multiple overlapping actions.

In future work will be looking into using spatial, as well as temporal relations in the system. We are also looking into trying out quantitative relations, so that a relation will not work on objects that are either too close, or too far away. We will also be looking into changing the output from a sub-model based on what data the search section has evaluated on. Finally we will be looking at speed improvements to the system so that the run time can be reduced.

Training Dataset	Number of runs	C(%)	A(%)
Snap No Noise	1	99.9	99.9
	4	99.8	99.8
Snap Noise	2	99.8	99.8
	1	99.8	96.6
	1	96.0	94.8
	1	96.0	93.3
Uno No Noise	1	99.7	99.7
	1	97.2	97.2
	1	94.0	92.0
	1	91.5	89.7
	1	88.8	88.8
Uno Noise	2	96.8	88.4
	1	95.1	95.1
	1	94.3	90.4
	1	88.5	87.1
Path No Noise	1	97.9	92.1
	1	97.9	89.3
	1	96.8	88.8
	1	95.8	90.0
	1	94.7	88.5
Path Noise	1	93.6	85.8
	1	92.6	85.2
	1	91.5	82.4
	1	90.8	83.3
	1	90.1	80.7

Figure 6. This figure shows the results for the snap, uno and path datasets. The number of runs column shows for each training set how many runs got the same coverage, and accuracy scores.

REFERENCES

[1] James Allen, 'Maintaining knowledge about temporal intervals', *Communications of the ACM*, **26**, 832–843, (1983).

[2] A. Bennett and D. Magee, 'Learning sets of sub-models for spatio-temporal prediction', in *Proceedings of AI-2007, the Twenty-seventh SGAI International Conference on Innovate Techniques and Applications of Artificial Intelligence*, pp. 123–136, (2007).

[3] Matthew Brand, 'Pattern discovery via entropy minimization', in *Artificial Intelligence and Statistics*, (1998).

[4] Simon Colton, Alan Bundy, and Toby Walsh, 'Automatic identification of mathematical concepts', in *International Conference on Machine Learning*, (2000).

[5] R. Etxeberria, P. Larranaga, and J.M. Picaza, 'Analysis of the behaviour of genetic algorithms when learning bayesian network structure from data', *Pattern Recognition Letters*, **13**, 1269–1273, (1997).

[6] Nir Friedman and Daphne Koller, 'Being bayesian about network structure', *Machine Learning*, **50**, 95–126, (2003).

[7] Aphrodite Galata, Neil Johnson, and David Hogg, 'Learning behaviour models of human activities', in *British Machine Vision Conference (BMVC)*, (1999).

[8] Kristen Grauman and Trevor Darrell, 'The pyramid match kernel:discriminative classication with sets of image features', in *International Conference on Computer Vision*, (2005).

[9] R Haenni, 'Towards a unifying theory of logical and probabilistic rea-

soning', in *International Symposium on Imprecise Probabilities and Their Applications*, pp. 193–202, (2005).

[10] John Koza, *Genetic Programming*, MIT Press, 1992.

[11] John Koza, *Genetic Programming II*, MIT Press, 1994.

[12] Philippe Leray and Olivier Francios, 'Bayesian network structural learning and incomplete data', in *Adaptive Knowledge Representation and Reasoning*, (2005).

[13] David Montana, 'Strongly typed genetic programming', in *Evolutionary Computation*, (1995).

[14] S.H. Muggleton and J. Firth, 'CProgol4.4: a tutorial introduction', in *Relational Data Mining*, 160–188, Springer-Verlag, (2001).

[15] Chris Needham, Paulo Santos, Derek Magee, Vincent Devin, David Hogg, and Anthony Cohn, 'Protocols from perceptual observations', *Artificial Intelligence*, **167**, 103–136, (2005).

[16] N. J. Nilsson, 'Probabilistic logic', *Artificial Intelligence*, **28**, 71–87, (1986).

[17] Jeffrey Mark Siskind, 'Grounding the lexical semantics of verbs in visual perception using force dynamics and event logic', *Articial Intelligence Research*, **15**, 31–90, (2000).