Detecting Sensor Dependencies for Building Complementary Model Ensembles

Alexander Dockhorn, Rudolf Kruse

Fakultät für Informatik, Otto von Guericke Universität Magdeburg
Universitätsplatz 2, 39106 Magdeburg, Germany
E-Mail: {alexander.dockhorn, rudolf.kruse}@ovgu.de

Abstract

Forward Model Approximation is a recently developed method for learning how to play unknown games. In this method game state transitions are analyzed with the target of predicting the result of the agent's actions. Instead of using a single complex model, complementary model ensembles can be used. This does not affect the accuracy in case each simple model focuses on independent components of the game. However, it is an open question how to retrieve these independent components without domain knowledge of the game being studied. In this work we explore the usage of bayesian belief network structure learning algorithms for the detection of sensor dependencies. Multiple structure learning algorithms are evaluated in a case-study involving two games of the GVGAI framework. Our results indicate that the analyzed algorithms are able to detect necessary dependencies for generating an approximated forward model.

1 Introduction

Popular training processes for artificial intelligence in games are based on either studying expert play or training an agent from scratch using reinforcement learning. Both method classes can achieve impressive results, but the learning process often converges slowly and results in a model that is hard to interpret (e.g., Deep Neural Networks).

In contrast, the Forward Model Approximation framework [6] learns to describe relevant components of an unknown game by learning either a complex model or a set of complementary models for the prediction of future states. Each of those sub-models focuses on describing the behavior of a single game component. Aggregating the results of all sub-models lets us predict future game states with higher accuracy than using a single model describing the behavior of all game components, while maintaining interpretability of each sub-model. Furthermore, the approximated forward model can be used to apply simulation-based search methods, such as Monte Carlo Tree Search [4], which proved to be among the top-performing algorithms in automated game-playing [7, 8, 13].

Previous attempts for building complementary model ensembles for Forward Model Approximation were based on exploiting the limitations of game description languages [9], such as the video game description language [14]. These description languages describe games as a set of distinct game components with limited interactions. On the one hand, knowing those limitations lets us create a conditionalized set of observations for the prediction of each distinct game component. This filtering of the data set's attributes reduces the amount of data for training each sub-model, speeds up the learning process, and generally assists in achieving a higher accuracy with the resulting aggregated model. On the other hand, knowing the limitations between variable interactions is not always ensured, which is why alternative methods for detecting relations between game components and available game observations need to be found.

In this work we study the application of bayesian belief network structure learning algorithms to detect dependencies among game objects and the available sensory information. These dependencies can further be used to create conditionalized databases for the model construction without the need of knowing the game's description. This will be the first step for creating an autonomous learning approach for modeling unknown games with the help of Forward Model Approximation.

In Section 2 we will briefly review general game playing. Section 3 presents a summary of the Forward Model Approximation framework and its benefits and drawbacks. To overcome the latter we study structure learning algorithms and their application to data that has been collected by studying games of the GVGAI framework in Sections 4 and 5. In Section 6 we present the results of our case-study and a discussion of the usability of applied algorithms for detecting sensor dependencies. We conclude our work in Section 7 in which we shortly summarize implications of this case-study and provide an outlook for future work.

2 General Game Playing

General game playing poses the problem of creating an agent that is capable to learn how to successfully play multiple games. Here, games are represented as the combination of an interactive environment and a controllable agent that interacts with it based on a set of pre-defined actions. Each interaction can modify the environment and, therefore, result in a new state. After an interaction the agent receives a response in form of a numerical reward and the updated state.

The environment can be represented as a probability distribution:

$$Pr\{R_{t+1} = r, S_{t+1} = s' | S_0, A_0, R_1, ..., S_{t-1}, A_{t-1}, R_t, S_t, A_t\} \quad (1)$$

where S_t represents the state of the environment at time step t, A_t is the action that the agent chose at time t, and R_{t+1} is the reward received as the environment's response. Thus, the future reward and the future state depend on all previous interactions between the environment and the agent. Hence, the complexity of this probability distribution grows exponentially over time.

For this reason, general game analysis is often restricted to environments that fulfill the Markov Property [18], in which the environment's response depends only on the current state and the chosen action. This reduces the probability distribution to:

$$Pr\{S_{t+1} = s', R_{t+1} = r | S_t = s, A_t = a\} \quad (2)$$

The success of an agent and its related policy π can be measured by the return G. The return measures the accumulated reward till the end of an episode.

$$G_t = R_{t+1} + R_{t+2} + R_{t+3} + ... + R_T$$

If either the episodes are non-ending or the agent needs to value immediate rewards higher than rewards received at a later point of time, the discounted return is used.

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{T} \gamma^k R_{t+k+1}$$

The parameter $\gamma \in [0, 1]$ is the discount rate.

Reinforcement learning techniques try to estimate the expected return q given a state-action pair and choose their action accordingly.

$$\begin{aligned} q_\pi(s,a) &= \mathbb{E}_\pi\left[G_t | S_t = s, A_t = a\right] \\ &= \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | S_t = s, A_t = a\right] \end{aligned} \tag{3}$$

For every game we search for an optimal policy π that maximizes the expected return of the agent. Given the expected return, the optimal action can be determined by:

$$\pi'(s) = \arg\max_a q_\pi(s,a) \tag{4}$$

3 Forward Model Approximation

The General Video Game AI (GVGAI) competition [13] offers two tracks, which focus on different aspects of general game playing. In the single-player planning track agents can use a forward model to predict the outcome of an action sequence. Simulation-based search algorithms such as Monte Carlo Tree Search [4] use this forward model to choose an action sequence that maximizes their expected score. Agents of the single-player learning track cannot access such a model, but can learn from repeated interactions with the environment.

While simulation-based search algorithms show great performance in the planning track, they cannot be applied without a forward model. Therefore, the missing forward model led to multiple submitted agents using reinforcement learning techniques, which learn the value of state-action pairs by playing the game many times. Recently, we introduced Forward Model Approximation [6] as a different solution to the special requirements of the learning track. In Forward Model Approximation we model the changes of the observed game state after applying an action using either a single or a set of classifiers. In case the agent uses a single classifier it tries to predict the future state based on the information on the current state and the action to be applied. Because this approach can result in a very complex classifier, we can also split it into multiple sub-models, of which each of these classifiers predicts the behavior of a single game entity or changes of a single observable value.

Both approaches become feasible in case the Markov Property holds, whereupon, the game state does not depend on all previous interactions, but only on the previous game state and the applied action. The transition to the future state can then be modeled using either a deterministic or a probabilistic model. In contrast to reinforcement learning approaches, learning an approximated forward model can be done using few iterations, which can later be used during simulation-based search. In case the model predicts the game's future states good enough, similar performance to the single player planning track can be achieved.

However, learning the approximated forward model is a challenging task due to the availability of a lot of sensor values, which might or might not have an influence on the different components of the game. Additionally, special attention should be paid on efficiency of the model building process to ensure short learning times. In a previous work [9] we explained how a complex forward model can be split in multiple sub-models to achieve not only higher accuracy, but also reduce the time spent for model construction. This approach is often justified due to independent components of a game being studied. A solution for model splitting exploits specific restrictions of the video game definition language [14] to restrict the sub-model learning process [9].

Despite the advantages of this method, we see two major constraints:

1. **Assumptions on the framework:** The split into multiple sub-models is based on the type definition of the video game definition language. This type definition lets the agent detect similar behaving entities using a type identifier. It cannot be assumed that this information is accessible in games outside of the GVGAI framework. Therefore, a natural extension would be to detect similar entities based on their visual representation or their behavior at run-time.

2. **Limitation to local influence factors:** Each of the created sub-models assumes independence of global influence factors, such that only entities close to the modeled entity are taken into account during the model building process. This is sufficient to detect interactions between neighboring entities. However, it is not able to model interactions between entities that are farther away, e.g., a switch opening a door which is far away cannot be represented.

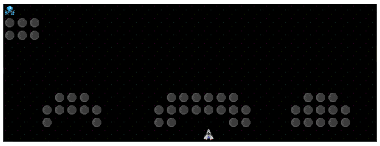
In this work we want to study methods for automatizing the filtering of data sets to remove these assumptions.

4 Structure Learning of Bayesian Belief Nets

Studying the dependence of variables among each other can be implemented by learning the structure of a bayesian belief net using the observed data set. A bayesian belief net is a directed acyclic graph, denoted by $\mathcal{G}$, with parameters Θ. Structure learning approaches try to find the DAG $\mathcal{G}$ that encodes the dependence structure of the data (see [2, 11] for a general description of bayesian belief networks). We differentiate three categories of structure learning algorithms, namely score-based, constraint-based, and hybrid approaches.

Scoring based approaches choose the best performing belief net structure over a set of candidate structures by measuring the goodness of fit between the structures implied probability distribution and the observed data. However, finding a belief net structure over a given set of variables using exhaustive search is super-exponential in the number of variables. Due to the high number of sensors and their variability of values in games of the GVGAI framework, this approach is infeasible. For this reason, local search algorithms restrict the search in each iteration to a subspace of network structures. For example the Hill Climbing (*hc*) algorithm [5, 10] modifies the current structure by either adding, removing, or reversing an edge. As a result of each possible modification the given graph structure is rated using a network score, such as the Bayesian Information Criterion [15]. The greedy *hc* algorithm is known to get trapped in local optima. Tabu-Search (*tabu*) [3] maintains a tabu list of previously performed modifications to avoid getting stuck.

Constraint-based approaches use conditional independence tests to learn the dependence structure of the attributes given the observed sample. One example is the Grow-Shrink (*gs*) algorithm [12], which first uses markov blankets to fix an undirected structure of the graph and then orients edges, removes existing cycles, and propagates directions to yet undirected edges. The semi-interleaved Hiton Parents and Children (*si-hiton-pc*) algorithm [1] learns local causal and markov blanket relationships. It is especially suited for large attribute sets with only limited data available. Another algorithm, which was developed with large attribute sets in mind, is the Max-Min Parents and Children (*mmpc*) algorithm [19]. Similar to the other two algorithms *mmpc* creates an undirected graph structure by detecting all possible parents and children per node.


(a) the *aliens* game

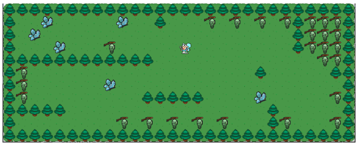

(b) the *butterflies* game

Figure 1: GVGAI games used for evaluation

Hybrid algorithms combine a restriction and a maximization phase. During the restriction phase the search space is reduced to a subset of possible DAG structures, e.g., by searching for all pairs of connected nodes. In the maximization phase a score-based algorithm is used to direct the edges of the determined graph structure. The Max-Min Hill-Climbing (*mmhc*) algorithm [20] uses *mmpc* to restrict the network structure and uses the *hc* algorithm to find the best configuration for all edge directions. A more general framework is the 2-phase Restricted Maximization (*rsmax2*), which lets the user choose the settings for both phases independently [16, 17].

In this work we will test the algorithms mentioned above to learn the structure of a belief net on the basis of all observable variables in the GVGAI framework. Using this approach we want to identify dependent and independent components in the belief net structure. While independent components help us to reduce the complexity of sub-models in the approximated forward model, dependent components point out interactions between variables that should not be ignored. These interactions can either be included in a sub-model building process or need to be added during the aggregation process.

5 Building a Data Set for GVGAI Games

We evaluate our process using two exemplary games of the GVGAI competition to compare the results of different structure learning algorithms.

In our first game *aliens* (see Figure 1a) the player steers a small space-ship at the bottom of the screen. In the boundaries of the screen it can either move left or right. The spaceship can also shoot to create a bullet over the player's avatar, which is moving to the top of the screen, where it

Table 1: Framework specific sensor values stored during the play session.

Player Sensors	NPC Sensors	Game State Sensors
• X and Y pos • X and Y grid pos • X and Y pos change • 8-directional neighborhood • selected action	• X and Y pos • X and Y grid pos • X and Y pos change • 8-directional neighborhood • previous X and Y position change • delay since last movement	• game tick • destroyed/created instances of type x • score change • win/lose/continuing

will be destroyed. Only one bullet can exist at a time. At the top of the screen multiple alien spaceships will move from left to right. In case an alien reaches the end of its row, instead of leaving the screen it will move down one row and change its flying direction. This process repeats until it reaches the players row in which case the player loses the game by touching the alien. This can only be avoided by shooting the alien spaceships before they reach the bottom row. Further, multiple boulders can be destroyed by shooting them. The player wins the game in case all aliens are destroyed, but loses if its own spaceship is hit by an alien.

The second game *butterflies* (see Figure 1b) tasks the player to catch all the butterflies flying around. These butterflies fly at random and can free more butterflies in case they are touching one of the cocoons. The player loses the game once all cocoons were destroyed. This can only be avoided by catching all the butterflies before they destroy all the cocoons.

Both games were played by a random agent not knowing any of the game's mechanics. These games were selected, because they are two of the few games a random agent might be able to win. During the play sessions we stored the sensor values listed in Table 1.

6 Evaluation of Structure Learning Algorithms

For the evaluation we created a data set of playthroughs using the 2017 Java version of the GVGAI framework. Our evaluation of structure

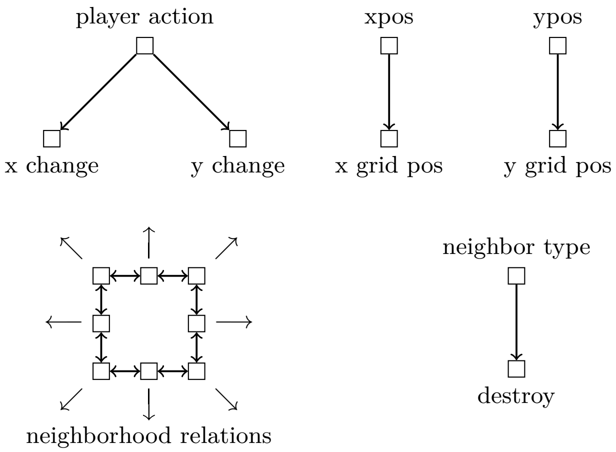


Figure 2: Retrieved movement specific dependence structures; (top left) the player's action and movement; (top right) position and grid position; (bottom left) neighborhood relations; (bottom right) threat of being destroyed by neighboring object

learning algorithms was performed using the *bnlearn* R-package [16]. During the preprocessing phase we nominalized all attributes even if they are on a numerical scale. This was done to ensure that all algorithms can be applied equally. The data set consists of the listed attributes (see Table 1) for every observable game element (the player and all non-player elements) as well as the general game state information for every game tick. In case an element cannot be observed during a game tick the value of all its attributes is set to *NA*.

Depending on the number of non-player elements (like butterflies or alien spaceships) the dataset contains more than 500 attributes. Attributes that only depict a single value over the time of a complete playthrough are removed during the preprocessing phase. We additionally deleted the observations of all objects that never change their position (X change, Y Change), are present from the beginning of the game, and are never removed during the playthrough. This was done to exclude background objects, which are observed and reported, but do never influence any of the game's components. All remaining attributes are used for learning the structure of a bayesian belief net.

Table 2: Summary of learned structures for the game; #E: number of edges, #V: number of vertices, #C: number of connected components; *the algorithms *gs*, *mmpc* and *si-hiton-pc* return an undirected graph, the listed number of edges is the number of undirected edges

algorithm	*alien*			*butterflies*		
	#E	#V	#C	#E	#V	#C
hc	361	370	9	339	318	2
tabu	361	370	9	343	318	1
gs	13*	26	13	28*	57	27
mmpc	21*	42	21	40*	76	36
si-hiton-pc	93*	184	91	148*	237	95
rsmax2 (gs, hc/tabu)	6	12	6	30	57	27
rsmax2 (mmpc, hc/tabu)	14	28	14	40	76	36
rsmax2 (si-hiton-pc, hc/tabu)	92	182	90	139	230	94

We qualitatively compare the structure learning algorithms *hc, tabu* (score-based), *gc, mmpc, si-hiton-pc* (constraints-based), and *rsmax2* for all combinations of mentioned score-based and constraint-based algorithms. Interesting dependency structures are shown in Figure 2. They highlight game specific dependencies that proved useful in previously generated approximated forward models, such as modeling the player's movement, understanding the dependency of an object's neighborhood sensors, and predicting the destruction of an object. The following sections present the results of processing data of the *aliens* and *butterflies* game.

6.1 Evaluation of the Game *Aliens*

The analysis of the game *aliens* is based on a single random playthrough, which consists of 715 observed game ticks and a total of 405 unfiltered attributes. Table 2 summarizes the resulting graph structures of applied structure learning algorithms.

The *hc* and the *tabu* algorithm return the same graph, which nearly includes all attributes and only a few components. Its biggest component consists of 293 vertices and includes movement, neighborhood, and destroy relations of nearly all aliens and alien shots. Even if unpractical, such a single big component is justified by the simultaneous movement of all alien objects. For this reason, the algorithm cannot detect that these objects act independently from each other. One of the smaller components including only 6 vertices represents the player's movement, which in this game is independent of other game objects. Remaining components include all attributes related to a single shot object, which did not collide with any other game object. For this reason, the observed shot was independent of all other observed game objects.

The *gs*, *mmpc*, *rsmax2(gs,hc/tabu)*, and *rsmax2(mmpc,hc/tabu)* algorithms all fail in detecting most attribute dependencies. Their resulting graphs only consist of a few edges correctly connecting the position and the grid position attributes of some instances. Additionally the result of the *rsmax2* algorithm only depends on the chosen restriction algorithm, as switching from *hc* to *tabu* did not change the resulting graph.

Applying the *si-hiton-pc* and *rsmax2(si-hiton-pc, hc/tabu)* returns a graph with many small communities, which represent characteristics of the game, but do not include all related attributes. These communities include parts of the neighborhood relation or movement based attribute interactions, but overall failed to detect necessary dependencies.

6.2 Evaluation of the Game *Butterflies*

In the second part of our case study we analyze the *butterflies* game. Our data set is also based on a single random playthrough, which consists of 667 observed game ticks and a total of 334 unfiltered attributes. Table 2 summarizes the resulting graph structures of applied structure learning algorithms.

As it was the case in the previous game, both the *hc* and the *tabu* algorithm return nearly similar graphs. The resulting graph structure includes all game object dependencies in a single component. In the case of the *butterflies* game, this is overly specific, since many of the game's objects act independent from each other. However, in case a butterfly comes into contact with a cocoon, it will destroy it and spawn a new butterfly. This interaction can be found in a specific playthrough, but might not appear in the next one. Currently, it is not possible to relate sensor values from differing playthrough, such that it is impossible to render interacting objects independent from each other.

The *gs* and *mmpc* algorithms as well as their *rsmax2* counterparts tend to form very small groups of attributes. Once again the output of the *rsmax2* algorithm only depends on the chosen restriction phase. The resulting graphs include parts of the neighborhood dependencies. Movement and destroy interactions were not included.

si-hiton-pc, *rsmax2(si-hiton-pc, hc)*, and *rsmax2(si-hiton-pc, tabu)* all result in similar graph structures including 95 connected components. These components model neighborhood interactions, but rarely connect attributes of differing objects.

6.3 Discussion

The results of our two case studies showed that the applied algorithms largely vary in the resulting graph structures. Generally, the constraint based algorithms *gs* and *mmpc* and their hybrid counterparts created very sparse graph structures. The generated structures map important attribute dependencies of the game, but do so very unreliably. Therefore, they are unsuited for a general framework for detecting object dependencies.

The tested score-based approaches (*hc* and *tabu*) as well as the constraint-based *si-hiton-pc* algorithm detected dependence structures on a reliable basis. The *hc* and *tabu* algorithms result in graphs with higher density, due to connections in between attributes of different game objects. These connected game objects interact only very rarely, therefore, learning a model on the returned graph structure may be over-specific to the single observed playthrough. In return, the *si-hiton-pc* algorithm and the *rsmax2* algorithm using *si-hiton-pc* during its restriction phase both detect most necessary attribute dependencies for modeling the behavior of the games' objects. However, inter-object attribute dependencies are not taken into account.

Using the generated graph structure of *si-hiton-pc* for the generation of an approximated forward model will result in a large number of simple models. The results of these models need to be aggregated and will rarely result in mistakes in case of interactions between multiple objects exist. Higher accuracy can be achieved by using the graph structures of the *hc* or the *tabu* algorithm. Those models will be able to achieve a higher accuracy at the cost of a higher complexity.

7 Conclusion and Future Work

In this work we studied the suitability of belief net structure induction algorithms for detecting object dependencies in games of the GVGAI framework. Out of the tested algorithms score-based algorithms such as *hc* and *tabu* seem to detect the most object dependencies. Even rare interactions between objects can be modeled using attribute sets of the resulting graph structures. Applying the *si-hiton-pc* or *rsmax2(si-hiton-pc, hc/tabu)* algorithm results in graphs with reduced density. Forward Model Approximation might benefit from this by a reduced complexity for each sub-model, which may also reduce the model's accuracy.

Our small case study showed the applicability of belief net structure induction algorithms for an automated analysis of dependency structures. Forward Model Approximation will benefit from this by allowing a divide of the agent's sensor values into independent sets. For each of these sets a simple model can be trained. Automatizing the division into multiple sets of independent sensor values makes it possible to apply Forward Model Approximation without information about the game's attributes' dependency structure.

One of the next steps will be to incorporate the structure learning algorithm to Forward Model Approximation. Thus, creating a unified framework for the analysis of unknown games without further domain knowledge. For this purpose, we will need to study the generalizability of the studied structure learning algorithms. Extending our case-study will provide further insights in characteristics of games and how they influence the applicability of varying structure learning algorithms.

References

[1] Constantin F Aliferis, Alexander Statnikov, Subramani Mani, and Xenofon D Koutsoukos. Local Causal and Markov Blanket Induction for Causal Discovery and Feature Selection for Classification Part I: Algorithms and Empirical Evaluation Ioannis Tsamardinos. *Journal of Machine Learning Research*, 11(March):171–234, 2010.

[2] Christian Borgelt, Matthias Steinbrecher, and Rudolf Kruse. *Graphical Models.* John Wiley & Sons, Ltd, Chichester, UK, August 2009.

[3] R.R. Bouckaert. *Bayesian Belief Networks: from Construction to Inference.* PhD thesis, Utrecht, Netherlands, 1995.

[4] Cameron B. Browne, Edward Powley, Daniel Whitehouse, Simon M. Lucas, Peter I. Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton. A Survey of Monte Carlo Tree Search Methods. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(1):1–43, March 2012.

[5] Gregory F Cooper and Edward Herskovits. A Bayesian method for the induction of probabilistic networks from data. *Machine Learning*, 9(4):309–347, October 1992.

[6] Alexander Dockhorn and Daan Apeldoorn. Forward Model Approximation for General Video Game Learning. In *2018 IEEE Conference on Computational Intelligence and Games, CIG 2018*, 2018.

[7] Alexander Dockhorn, Christoph Doell, Matthias Hewelt, and Rudolf Kruse. A decision heuristic for Monte Carlo tree search doppelkopf agents. In *2017 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 1–8. IEEE, November 2017.

[8] Alexander Dockhorn, Max Frick, Ünal Akkaya, and Rudolf Kruse. Predicting opponent moves for improving hearthstone ai. In Jesús Medina, Manuel Ojeda-Aciego, José Luis Verdegay, David A. Pelta, Inma P. Cabrera, Bernadette Bouchon-Meunier, and Ronald R. Yager, editors, *Information Processing and Management of Uncertainty in Knowledge-Based Systems. Theory and Foundations*, pages 621–632, Cham, 2018. Springer International Publishing.

[9] Alexander Dockhorn, Tim Tippelt, and Rudolf Kruse. Model Decomposition for Forward Model Approximation. In *2018 IEEE Symposium Series on Computational Intelligence (SSCI)*, 2018.

[10] José A. Gámez, Juan L. Mateo, and José M. Puerta. Learning Bayesian networks by hill climbing: efficient methods based on progressive restriction of the neighborhood. *Data Mining and Knowledge Discovery*, 22(1-2):106–148, January 2011.

[11] Rudolf Kruse, Christian Borgelt, Christian Braune, Sanaz Mostaghim, and Matthias Steinbrecher. *Computational Intelligence.* Texts in Computer Science. Springer London, London, 2nd editio edition, 2016.

[12] Dimitris Margaritis, Sebastian Thrun, Christos Faloutsos, Andrew W Moore, and Gregory F Cooper. *Learning Bayesian Network Model Structure from Data.* PhD thesis, Carnegie Mellon University, 2003.

[13] Diego Perez-Liebana, Spyridon Samothrakis, Julian Togelius, Tom Schaul, Simon M. Lucas, Adrien Couetoux, Jerry Lee, Chong U. Lim, and Tommy Thompson. The 2014 General Video Game Playing Competition. *IEEE Transactions on Computational Intelligence and AI in Games*, 8(3):229–243, 2016.

[14] Tom Schaul. An extensible description language for video games. *IEEE Transactions on Computational Intelligence and AI in Games*, 6(4):325–331, 2014.

[15] Gideon Schwarz. Estimating the Dimension of a Model. *The Annals of Statistics*, 6(2):461–464, March 1978.

[16] Marco Scutari. Learning bayesian networks with the bnlearn R package. *Journal of Statistical Software*, 35(3):1–22, 2010.

[17] Marco Scutari and Jean-Baptiste Denis. *Bayesian Networks with Examples in R.* Chapman and Hall, Boca Raton, 2014. ISBN 978-1-4822-2558-7, 978-1-4822-2560-0.

[18] Publisher Taylor. The Oxford Dictionary of Statistical Terms. *Technometrics*, 46(2):266–266, May 2004.

[19] Ioannis Tsamardinos, Constantin F. Aliferis, and Alexander Statnikov. Time and sample efficient discovery of Markov blankets and direct causal relations. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '03*, page 673, New York, New York, USA, 2003. ACM Press.

[20] Ioannis Tsamardinos, Laura E. Brown, and Constantin F. Aliferis. The max-min hill-climbing Bayesian network structure learning algorithm. *Machine Learning*, 65(1):31–78, 2006.