A Dynamic Bayesian Network Framework for Learning from Observation

Santiago Ontañón[1], José Luis Montaña[2], and Avelino J. Gonzalez[3]

[1] Drexel University, Philadelphia, PA, USA 19104
santi@cs.drexel.edu
[2] University of Cantabria, Santander, Spain
montanjl@unican.es
[3] University of Central Florida, Orlando, FL, USA
gonzalez@ucf.edu

Abstract. Learning from Observation (a.k.a. learning from demonstration) studies how computers can learn to perform complex tasks by observing and thereafter imitating the performance of an expert. Most work on learning from observation assumes that the behavior to be learned can be expressed as a state-to-action mapping. However most behaviors of interest in real applications of learning from observation require remembering past states. We propose a Dynamic Bayesian Network approach to learning from observation that addresses such problem by assuming the existence of non-observable states.

1 Introduction

Learning by watching others do something is a natural and highly effective way for humans to learn. It is also an intuitive and highly promising avenue for machine learning. It might provide a way for machines to learn how to perform tasks in a more natural fashion. This form of learning is known as *Learning from Observation* (LfO). Works reported in the literature also refer to *learning from demonstration*, *learning by imitation*, *programming by demonstration*, or *apprenticeship learning*, as largely synonymous to LfO.

This paper presents a new framework for LfO, based on *Dynamic Bayesian Networks* [7], called LfODBN. While there has been much work on LfO in the past (for a recent overview, see [1]), most proposed approaches assume that the behavior to be learned can be represented as a situation-to-action mapping (a policy). This assumes that, in the behavior to be learned, the choice of actions depends only on the current observable state. However, most behaviors of interest in the real world do not satisfy this restriction. For example, if we were to teach a robot how to automatically drive a car, the robot will need to remember past information that is not part of the current observable state, such as what was the last speed limit sign seen.

In general, the problem is that when learning from observation, the learning agent can observe the state of the world and the actions executed by the demonstrator or expert, but not the internal mental state of the expert (e.g. her

C. Bielza et al. (Eds.): CAEPIA 2013, LNAI 8109, pp. 373–382, 2013.

memory). The LfODBN model presented in this paper takes into account that the expert has a non-observable internal state, and, under some restrictions, can learn such behaviors.

Work in learning from observation can be traced back to the early days of AI. For instance, Bauer [2] proposed in 1979 to learn programs from example executions, which basically amounts to learning strategies to perform abstract computations by demonstration. This form of learning was especially popular in robotics [8]. Modern work on the more general LfO subject came from Sammut et al [15] and Sidani [17]. Fernlund et al. [5] used learning from observation to build agents capable of driving a simulated automobile in a city environment. Pomerleau [13] developed the ALVINN system that trained neural networks from observation of a road-following automobile in the real world. Although the neural network approach to learning from observation has remained popular with contributions such as the work of Moriarty and Gonzalez [9], LfO has been explored in the context of many other learning paradigms such as reinforcement learning [16], *case-based reasoning* (CBR) [6,11], and *Inverse Reinforcement Learning* (IRL) [10]. These approaches, however, ignore the fact that the expert might have internal state. For example, IRL assumes the expert is solving a Markov Decision Process (MDP), and thus has no additional internal state other than the observed state. For IRL to be applicable to the general problem of LfO, it needs to consider partially observable MDPs (POMDP), to account for the lack of observability of the expert's state.

The remainder of this paper is organized as follows. Sections 2 and 3 present some background on dynamic Bayesian networks and learning from observation respectively. Then, Section 4 presents our LfODBN model. After that, Section 5 empirically evaluates the LfODBN in a synthetic benchmark.

2 Background

A Bayesian Network (BN) is a modeling tool that represents a collection of random variables and their conditional dependencies as a directed acyclic graph (DAG). In this paper, we are interested in a specific type of BNs called *Dynamic Bayesian Networks* (DBN) [7]. In a DBN, the variables of the network are divided into a series of identical time-slices. A time-slice contains the set of variables representing the state of the process that we are trying to model at a given instant of time. Variables in a time-slice can only have dependencies with variables in the same or previous time-slices. DBNs can be seen as graphical representations of *stochastic processes*, i.e. random processes that depend on time [12].

The most common example of a DBN is the Hidden Markov Model [14], or HMM. There are only two variables in each time slice t in an HMM. A hidden variable C_t, typically called the *state*, and an observable variable Y_t, typical called the *output*. The output Y_t only depends on the state C_t, and the state C_t only depends on the state in the previous time slice, C_{t-1} (except in the first time slice). Moreover, the conditional probabilities $p(C_t|C_{t-1})$ and $p(Y_t|C_t)$ are assumed to be independent of t.

Although HMMs are the best known DBN and have many applications, such as speech recognition [14], there are other well-studied DBNs such as *Input-Output Hiden Markov Models* (IOHMM) [3]. In an IOHMM, in addition to the state and the output, there is an observable input variable, X_t upon which both the state C_t and the output Y_t depend. In the remainder of this paper we will use the following convention: if X is a variable, then we will use a calligraphic $\mathcal{X}$ to denote the set of values it can take, and lower case to denote the specific values it takes, i.e. $x_t \in \mathcal{X}$.

3 Learning from Observation

The goal of learning from observation (LfO) is to automatically learn a behavior by observing an expert perform a given task. The main difference between LfO and standard supervised learning is that the goal is to learn a behavior that might vary over time, rather than approximating a static function. The basic elements in LfO are the following:

- There is an *environment* E.
- There is one *actor* (or trainer, expert, or demonstrator), who performs a task in the environment E.
- There is a *learning agent* A, whose goal is to learn how to achieve the task in the environment E by observing the actions performed by the actor.

In learning from observation, the learning agent A first observes one or several actors performing the task to be learned in the environment, and records their behavior in the form of *traces*, from where behavior is learned. Some learning from observation approaches assume that the learner also has access to a reward signal R. In our framework we will assume such reward signal is not available, and that the goal is thus to just imitate the actor.

Specifically, the behavior of an agent can be captured by three different variables: its perception of the environment, X, its unobservable internal mental state C, and the perceptible actions it executes, Y. We will define $\mathcal{I} = \mathcal{X} \times \mathcal{C} \times \mathcal{Y}$, and interpret the actor behavior as a stochastic process $I = \{I_1, ..., I_n, ...\}$, with state space $\mathcal{I}$. $I_t = (X_t, C_t, Y_t)$ is the random variable where X_t and Y_t represent respectively the input and output variables at time t, and C_t represents the internal state of the actor at time t. The observed behavior of an actor in a particular execution defines a *learning trace*: $LT = [(x_1, y_1), ..., (x_n, y_n)]$ where x_t and y_t represent the specific perception of the environment and action of the actor at time t. The pair of variables X_t and Y_t represent the *observation* of the learning agent A, i.e.: $O_t = (X_t, Y_t)$. Thus, for simplicity, we can write a learning trace as $LT = [o_1, ..., o_n]$.

We assume that the random variables X_t and Y_t are multidimensional variables that can be either continuous or discrete. In our framework, thus, the LfO problem reduces to estimating the unknown probability measure that governs the stochastic process, taking as input a data set of k trajectories $\{LT_j : 1 \leq j \leq k\}$ of the stochastic process I.

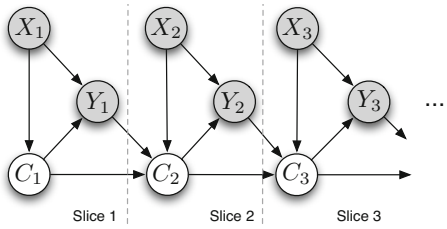

Fig. 1. The LfODBN Model. Grayed out variables are observable by the learning agent, white variables are hidden. X_t is the perception of the state, Y_t is the action, and C_t is the internal state of the agent.

As mentioned above, most work on LfO [1] assumes that the action Y_t depends exclusively on the state X_t (i.e. that the behavior is Markovian). Under this assumption, each of the entries (x_t, y_t) in a trace can be taken as individual examples in a supervised learning framework. Thus, if we assume that the action the expert executes at time t only depends on the perception at time t, then learning from observation is equivalent to supervised learning. However, in many real-life behaviors this assumption doesn't hold.

Consider the following example. When a driver in a highway sees a sign indicating the desired exit is approaching, the driver starts merging to the right lanes, even if she does not see the sign any more. Thus, the driver needs to remember that she has seen such sign (in her internal state C).

As a second example, imagine an agent wants to learn how to play Stratego by observation. Stratego is similar to Chess, but players do not see the types of the pieces of the opponent, only their locations. Thus, the perception of the state X_t contains only the locations of the pieces of the opponent (in addition to the player's piece locations and types). After certain movements, a player can temporally observe the type of one piece, and must remember this in order to exploit this information in the future. In this case, the internal state C_t of an actor should contain all the types of the opponent pieces observed up to time t.

The typical strategy to avoid this situation when designing a LfO system is to identify all of those aspects the expert has to remember, and include them in the set of input features. For example, we could add a variable to x_t representing "which is the last exit sign we saw in the highway". However, this requires manual "feature engineering", which is highly undesirable.

4 DBN-Based Learning from Observation

Using the DBN framework, we can represent the probability distribution of the stochastic process representing the behavior of an actor as the network shown in Figure 1, that we call the *LfODBN* model. The LfODBN model contains all the variables in LfO and their conditional dependencies (grayed out variables are observable, white variables are hidden). The internal state of the actor at time t, C_t, depends on the internal state at the previous instant of time, C_{t-1}, the

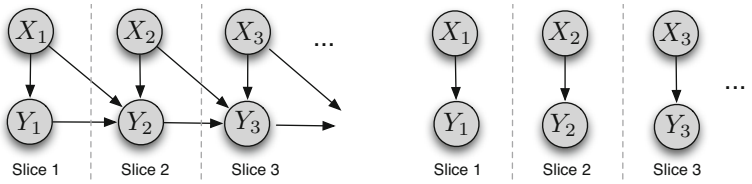

Fig. 2. Simplifications of the model in Figure 1 For assumptions 1 (right) and 2 (left)

previous action Y_{t-1} and of the current observation X_t. The action Y_t depends only on the current observation, X_t and the current internal state C_t.

Given the LfODBN model, if the learning agent wants to learn the behavior of the expert, it has to learn the dependencies between the variables C_t, X_t, and Y_t, i.e. it has to learn the following conditional probability distributions: $\rho(C_1)$, $\rho(Y_t|C_t, X_t)$, and $\rho(C_t|C_{t-1}, X_t, Y_{t-1})$. If the learning agent is able to infer the previous conditional probability distributions, it can replicate the behavior of the expert. In practice, the main difficulty is that the internal state variable C_t is not observable, which, although typically neglected in the LfO literature, plays a key role in many behaviors of interest.

Also, notice that the goal of LfO is just to learn to replicate the behavior (i.e. actions) of the actor. Thus, relations such as the dependency of X_t in X_{t-1} and Y_{t-1} (that captures the effect that actions have on the environment) are irrelevant. Those relations would be key, however, if the learning agent was learning a model of the world for planning purposes.

Let us now present three approaches to LfO based on making three different assumptions over the internal state of the actor C_t.

4.1 Assumption 1: No Internal State

The assumption that the expert has no internal state, i.e. that C_t is irrelevant, is equivalent to assuming the behavior is reactive, and thus the action Y_t only depends on the current observation (X_t). In this case, we can simplify the LfODBN model as shown on the right hand side of Figure 2. Under this assumption, we can just use standard supervised learning techniques to learn the conditional probability $\rho(Y_t|X_t)$.

In this approach, each entry in a learning trace can be treated independently, and any supervised learning algorithm such as decision trees, neural networks or SVMs can be used. This is the simplest approach to LfO, with the only drawback that it cannot learn any behavior that requires the agent to remember anything from past states. The next two approaches make less restrictive assumptions about the internal state of the expert, to alleviate this problem.

4.2 Assumption 2: Time Window

In this approach, we assume that the expert internal state is a time window memory that stores the last k observations (i.e., the current state X_t, and the

last $k-1$ observations $O_{t-1}, ..., O_{t-(k-1)}$). For example, if $k = 2$, the expert internal state is $C_t = (X_t, O_{t-1})$. Under this assumption we can reformulate the LfODBN model, as shown on the left hand side of Figure 2 for $k = 2$. Notice that given k, we can ignore C_t in the DBN model, and thus, we still have no hidden variables. In general, for any given k, the conditional probability that must be learned is: $\rho(Y_t|X_t, O_{t-1}, ..., O_{t-(k-1)})$.

In this approach, each subsequence of k entries in a learning trace can be treated independently as a training example, and we can still use supervised learning techniques. The main drawback of this approach is that, as k increases, the number of features in the training examples increases, and thus, the learning task becomes more complex.

4.3 Assumption 3: Finite Discrete Internal State

Using the time window assumption, it is possible to learn behaviors where the agent only needs to remember a fixed number of past states; however, in general, the agent might need to remember a past state that is arbitrarily far in the past.

In this more general assumption, we assume that the internal state of the expert is discrete and can take a finite amount l of different values. In this assumption, we need to consider the complete LfODBN model as shown in Figure 1. Under this assumption, the Expectation-Maximization (EM) algorithm [4] can be used to learn the parameters of the LfODBN.

A possible simplification assumes that the internal state C_t depends only on previous internal state C_{t-1} and observation X_t (i.e. that it does not depend on the past action). The resulting model corresponds to an *Input-Output Hidden Markov Model* (IOHMM), for which specialized algorithms are known [3].

5 Experimental Evaluation

This section presents an experimental validation of algorithms based on the three assumptions presented above, and compares them with other common LfO algorithms in the literature. Specifically, these experiments are designed to show that standard algorithms used in the literature of LfO (such as neural networks) can only learn a limited set of behaviors; algorithms based on the LfODBN model, however, make a less restrictive assumption on the internal state of the expert, and thus, can learn a wider range of behaviors.

The domain we used for our experiments simulates an automatic vacuum cleaner navigating a room, and removing dirt spots. The goal is to remove dirt spots in a grid map. For these experiments, all the obstacles are static, and the only moving object in the simulation is the vacuum cleaner. The simulation time is discreet, and at each time step, the vacuum cleaner can take one out of these 5 actions: up, down, left, right and stand still, with their intuitive effect (if the vacuum tries to move into an obstacle, the effect is equivalent to the stand still action). Actions are deterministic. Thus, the control variable Y can take 5 different values: $\{up, down, left, right, stand\}$.

The vacuum cleaner perceives the world through 8 different variables: two binary variables per direction (up, down, left, right), one of them identifying what can the vacuum cleaner see in each direction (dirt or obstacle), and the other determining whether the object being seen is close (touching) or far. For the experiments, we created a collection of 7 different maps, of different sizes, from 8x8 to 32x32 and with different configuration of obstacles and dirt (with between 2 to 8 dirt spots). We created several different experts to learn from:

RND: SmartRandom. This agent executes random actions, except if it sees dirt in one of the four directions, in which case it will move straight for it.

STR: SmartStraightLine. This agent picks a direction at random and moves in a straight line until collision. Then, it repeats its behavior. But if it sees dirt in one of the four directions, it will move straight for it.

ZZ: ZigZag. This agent moves in zig-zag: it moves to the right, until colliding, then moves down and starts moving to the left until colliding. When it cannot go down any further, it repeats the behavior, but going up, and so on.

SEQ: FixedSequence. This agent always repeats the same, fixed, sequence of actions (15 actions long). Once the sequence is over, it restarts from scratch.

EXP: SmartExplorer. This is a complex agent that remembers all the cells in which it has already been. With a high probability (0.75) it selects the action that will lead him closer to an unexplored cell. Once all the cells in the map have been explored, the agent stops. If it sees dirt in one of the four directions, it will move straight for the dirt. Notice that in order to perform this behavior, the agent needs to remember each cell it has visited before.

We generated a total of 35 learning traces (one per expert per map). Each learning trace is 1000 steps long. Therefore, the learning agents have 7 learning traces per expert. We compared the performance of the following algorithms:

Algorithms Making Assumption 1: We used NN (Neural Networks), widely used in the literature of LfO [13,9], and BN (Bayesian Networks), a direct implementation of the simplified Bayesian Network shown in Figure 2. Neural Networks in our experiments have one hidden layer with 10 nodes, and 5 output nodes (one per possible action).

Algorithms Making Assumption 2: We also experimented with two algorithms in this case: NNk2 (Neural Networks) and BNk2 (Bayesian Networks). For both algorithms, we used $k = 2$, i.e. they learn to predict the expert actions based on the current state of the world, and the state and action in the previous instant of time.

Algorithms Making Assumption 3: We experimented with using the EM algorithm[1] to learn the parameters of both our proposed LfODBN model (Figure 1), as well as an IOHMM. In both cases, we ran 20 iterations of EM, and limited the internal state to have 4 different values.

[1] Specifically, we used the EM implementation in the Matlab *Bayes Net Toolbox* using the *jtree_2TBN_inf_engine* inference engine.

Table 1. Output Evaluation of the different LfO algorithms consisting of the percentage of expert actions predicted correctly. The bottom row shows the average of all the other rows, showing that LfODBN obtains the highest accuracy overall.

	NN	BN	NNk2	BNk2	IOHMM	LfODBN
RND	**32.0**	**30.9**	**32.0**	**31.0**	**31.0**	**31.1**
STR	40.0	40.7	**85.1**	**84.8**	77.2	**84.3**
ZZ	41.3	40.9	73.7	**91.6**	65.2	83.4
SEQ	43.2	36.2	66.4	51.9	85.8	**88.2**
EXP	48.4	49.3	**79.1**	77.6	65.3	**79.3**
Avg.	41.0	39.6	67.26	67.38	64.9	**72.3**

We evaluated the performance of the algorithms by measuring their accuracy in predicting the actions executed by the experts. For this purpose, we performed a leave-one-out evaluation, where agents learned from 6 learning traces, and were asked to predict the actions in the 7th, test trace. Specifically, given a model M learned by one of the learning algorithms, and a test trace LT containing n entries, the predictive accuracy $Acc(M, LT)$ was measured as follows:

$$P(M, LT, t) = \begin{cases} 1 & \text{if } M(x_t, [o_{t-1}, ..., o_1]) = y_t \\ 0 & \text{otherwise} \end{cases}$$

$$Acc(M, LT) = \frac{1}{n} \sum_{t=1...n} P(M, LT, t)$$

where $M(x_t, [o_{t-1}, ..., o_1])$ represents the action predicted by the model M given the observation at time t, and the entire subtrace from time 1 to time $t-1$. Since our traces have 1000 entries each, and we had 7 traces per expert, each reported result is the average of 7000 predictions.

Table 1 shows the predictive accuracy of each learning algorithm when learning from each of the experts. The best results for each expert are highlighted in bold (when more than one learning agent achieved statistically undistinguishable results, all of them are highlighted in bold). The easiest behavior to learn is the SmartRandom (RND) expert. All the learning agents were capable to perfectly learning this behavior. Notice that, even if the behavior is perfectly learned, they can only predict about a 31% of the actions of this expert, since the behavior of the expert involved randomness.

Next in difficulty is the SmartStraightLine (STR) expert. For this behavior, agents need to remember what was the last direction in which they moved. Thus, learning agents using assumption 1 (NN and BN) could simply not learn this behavior. All the other learning agents could learn this behavior perfectly (except IOHMM, which learned a pretty good approximation, but not exactly). The problem with IOHMM is that the relationship between Y_{t-1} and C_t is not present in the DBN, and thus, it has troubles learning behaviors that depend on the previous action. Again, no agent reached a 100% of prediction accuracy, since the expert would pick a random direction each time it hit a wall.

The ZigZag (ZZ) agent is even harder to learn, since, in addition to the last direction of movement, the agent must remember whether it is currently traveling down or up. No learning algorithm was able to learn this properly. All the algorithms making assumptions 2 and 3 properly learned the left-to-right behavior (and thus the high accuracy of BNk2), but none was capable of learning when to move down or up when changing directions. This was expected for algorithms making assumption 2, however, algorithms making assumption 3 are, in principle, capable of learning this. GEM was not capable of learning this from the training data provided though, and ended up learning only an approximate behavior, with some mistakes.

The FixedSequence (SEQ) expert is complex to learn, since, in order to learn the fixed sequence of 15 moves, agents must internally remember in which of the 15 states of the sequence they are. By using the past action as a reference, algorithms making assumption 2 (NNk2, BNk2) could better learn this behavior better that agents making assumption 1 (NN, BN) (increasing the value of k all the way up to 15 should let agents using assumption 2 learn this behavior, but with prohibitive number of features). However, only agents making assumption 3 could learn a good enough approximation.

Finally, the SmartExplorer (EXP) expert is very hard to learn, since it involves remembering every cell that has been visited. None of the agents was able to learn this behavior. Some algorithms, like LfODBN, have a high predictive accuracy (79.3%) just because they appropriately learn the probability of the expert to stop (the expert stops after exploring the whole room), and then they can predict correctly that the expert will just issue the *stand* action till the end of the trace. The bottom row of Table 1 shows the average predictive accuracy for all the learning algorithms we experimented with, showing that LfODBN obtains the highest accuracy overall.

6 Conclusions

This paper has presented a model of Learning form Observation (LfO) based on Dynamic Bayesian Networks (DBN), called LfODBN. The main contribution of this model is that it makes explicit the need for accounting for the unobservable internal state of the expert when learning a behavior from observation.

Additionally, we proposed three different approaches to learn from observation, based on three different assumptions on the internal state of the expert: 1) assume the expert has no internal state, 2) assume the internal state of the expert is a memory of the last k states, and 3) assume the expert has a finite discrete internal state. Each of the three assumptions leads to a different collection of algorithms: the first two can be addressed with supervised learning algorithms, but the last requires a different learning approach (for which we propose to use DBN learning algorithms that account for hidden variables).

Our experimental results show that algorithms making different assumptions can learn different ranges of behaviors, and that supervised learning approaches to LfO are not enough to deal with the general form of the LfO problem.

As part of our future work, we want to explore further less restrictive assumptions over the internal state of the expert, that allow learning broader ranges of behaviors, while still being tractable. Finally, we would also study better evaluation metrics for LfO, since, as observed in this paper, traditional classification accuracy is not very representative of the performance of LfO algorithms.

Acknowledgements. This work is partially supported by spanish grant TIN2011-27479-C04-04.

References

1. Argall, B.D., Chernova, S., Veloso, M., Browning, B.: A survey of robot learning from demonstration. Robot. Auton. Syst. 57, 469–483 (2009)
2. Bauer, M.A.: Programming by examples. Artificial Intelligence 12(1), 1–21 (1979)
3. Bengio, Y., Frasconi, P.: Input/output hmms for sequence processing. IEEE Transactions on Neural Networks 7, 1231–1249 (1996)
4. Dempster, A.P., Laird, N.M., Rubin, D.B.: Maximum likelihood from incomplete data via the em algorithm. Journal of the Royal Statistical Society, Series B 39(1), 1–38 (1977)
5. Fernlund, H.K.G., Gonzalez, A.J., Georgiopoulos, M., DeMara, R.F.: Learning tactical human behavior through observation of human performance. IEEE Transactions on Systems, Man, and Cybernetics, Part B 36(1), 128–140 (2006)
6. Floyd, M.W., Esfandiari, B., Lam, K.: A case-based reasoning approach to imitating robocup players. In: Proceedings of the Twenty-First International Florida Artificial Intelligence Research Society (FLAIRS), pp. 251–256 (2008)
7. Ghahramani, Z.: Learning dynamic Bayesian networks. In: Caianiello, E.R. (ed.) Adaptive Processing of Sequences and Data Structures, International Summer School on Neural Networks. Tutorial Lectures, pp. 168–197. Springer, London (1998)
8. Lozano-Pérez, T.: Robot programming. Proceedings of IEEE 71, 821–841 (1983)
9. Moriarty, C.L., Gonzalez, A.J.: Learning human behavior from observation for gaming applications. In: FLAIRS Conference (2009)
10. Ng, A.Y., Russell, S.: Algorithms for Inverse Reinforcement Learning. In: in Proc. 17th International Conf. on Machine Learning, pp. 663–670 (2000)
11. Ontañón, S., Mishra, K., Sugandh, N., Ram, A.: On-line case-based planning. Computational Intelligence Journal 26(1), 84–119 (2010)
12. Papoulis, A., Pillai, S.U.: Probability, Random Variables, and Stochastic Processes. McGraw-Hill Series in Electrical and Computer Engineering. McGraw-Hill (2002)
13. Pomerleau, D.: Alvinn: An autonomous land vehicle in a neural network. In: Touretzky, D.S. (ed.) Advances in Neural Information Processing Systems, vol. 1. Morgan Kaufmann (1989)
14. Rabiner, L.R.: A tutorial on hidden markov models and selected applications in speech recognition. Proceedings of the IEEE, 257–286 (1989)
15. Sammut, C., Hurst, S., Kedzier, D., Michie, D.: Learning to fly. In: Proceedings of the Ninth International Workshop on Machine Learning (ML 1992), pp. 385–393 (1992)
16. Schaal, S.: Learning from demonstration. In: NIPS, pp. 1040–1046 (1996)
17. Sidani, T.: Automated Machine Learning from Observation of Simulation. Ph.D. thesis, University of Central Florida (1994)