Why Experimentation can be better than "Perfect Guidance"*

Tobias Scheffer
Technische Universität Berlin
Artificial Intelligence Group, FR 5-8
Franklinstr. 28/29
10587 Berlin, Germany
scheffer@cs.tu-berlin.de

Russell Greiner and **Christian Darken**
Siemens Corporate Research
Adaptive Signal and Information Processing
755 College Road East
Princeton, NJ 08540, USA
{greiner, darken}@scr.siemens.com

Abstract

Many problems correspond to the classical control task of determining the appropriate control action to take, given some (sequence of) observations. One standard approach to learning these control rules, called *behavior cloning*, involves watching a perfect operator operate a plant, and then trying to emulate its behavior. In the *experimental learning* approach, by contrast, the learner first guesses an initial operation-to-action policy and tries it out. If this policy performs suboptimally, the learner can modify it to produce a new policy, and recur. This paper discusses the relative effectiveness of these two approaches, especially in the presence of perceptual aliasing, showing in particular that the experimental learner can often learn more effectively than the cloning one.

1 INTRODUCTION

Many real-world tasks require controlling some type of "plant" (*e.g.*, factory [DB95], grinding tool [Bro93], robot [Kae93]); in each case, using information received from the sensors to decide on a control action — *e.g.*, using temperature and pressure readings to decide whether to open some valve. In general a control policy specifies the action to take given any current (history of) sensor readings; and a controller is optimal if it minimizes some user-specified cost function. This task is complicated by several factors [WB91], including *perceptual aliasing* (*i.e.*, several different states may produce the same observations, which means the action appropriate for an observation one time may be problematic another time); and *stochasticity* in both observations (*i.e.*, a given state may produce different observations in different times) and action (*i.e.*, an action may not deterministically map one state to another).

Our task is to *learn* an optimal controller, based on information available about the plant. In the "behavioral cloning" approach [SHKM92, SMB95, Kha96], a learner L_B watches an "optimal" operator (*i.e.*, an operator that always performs the best possible action) operate a plant, and then tries to emulate this operator. Obvious examples include learning to fly an airplane [SHKM92], to control a satellite [MW95], to operate a grinding machine [Bro93] or to play WUMPUS effectively [RN95]; related techniques are also used to acquire expert-level knowledge [SCG91]. Of course, this "to-be-mimicked" operator can be a problem solver that generates optimal actions for a set of sample states [MW95, BSW96].

Alternatively, an "experimental learner" L_E would initially guess a control policy and test it out. This learner may then tweak the control policy, producing a new one that is (it hopes) superior, and so forth. This is the underlying principle of many *reinforcement learning* schemes [MC68, BSA83].

This paper compares these two classes of learners. Section 2 first provides the necessary definitions, defining in particular the "control" performance task, the basic learning task, and the two learning paradigms. Section 3 then describes when each paradigm should work most effectively. Section 4 provides a set of empirical results that support our claims.

*Appeared in *Proc. ICML, 97*

2 DEFINITIONS: PLANT, STRATEGY, LEARNING

Following standard control theory terminology, we view a "plant" as a deterministic finite automaton (DFA) $\Xi = \langle X, E, Y, U, \ell_{XY}, \ell_E, \ell_C, x_{init} \rangle$ whose nodes $x \in X$ are each labeled with both (1) an observable $y = \ell_{XY}(x) \in Y$ using the $\ell_{XY} : X \mapsto Y$ mapping; and (2) a non-negative cost using the $\ell_C : X \mapsto \Re^{\geq 0}$ mapping. Each directed edge $e \in E \subset X^2$ is labeled with an action $u = \ell_E(e) \in U$.

In general, if the plant is in state x, it will "emit" an observation $y = \ell_{XY}(x)$, and impose a cost $c = \ell_C(x)$. If the operator then "steers" the plant with the action $u \in U$, the plant will proceed along the edge $e = \langle x, x' \rangle \in E$ whose label is $\ell_E(e) = u$, to the new state x'. Here, we will write $x \mapsto^u x'$. The plant starts in state $x_{init} \in X$; in a slight abuse of terminology, we can view this x_{init} as a distribution over states in X.

We will also consider *probabilistic* finite automata, PFAs, which also require a probability mapping over edges $\ell_P : E \mapsto [0, 1]$, with the understanding that the action u maps x to x' with probability $p = \ell_P(\langle x, x' \rangle)$, where $\ell_E(\langle x, x' \rangle) = u$. Here, we will write $x \mapsto^{u,p} x'$. Similarly ℓ_{XY} need not be a function in a PFA, but can instead stochastically present some $y \in Y$ observable when the automaton is in state X.

Strategy: A "strategy" is a (possibly stochastic) function $op : Y \mapsto U$ that maps each observation into an action, in an attempt to "steer" the plant through a *trajectory* $\vec{x} = \langle x_1, x_2, \ldots, x_m \rangle$ where x_1 is drawn from x_{init} and each $x_i \mapsto^{u_i} x_{i+1}$ where $u_i = op(y_i)$ is the action prescribed by the strategy after it observes the output $y_i = \ell_{XY}(x_i)$. The cost of a trajectory $c(\vec{x}) = \frac{1}{m} \sum_i \ell_c(x_i)$ is the average cost of the states encountered; and we say the expected cost of a strategy $c(op)$ is the expected cost of the trajectory produced, over all encountered starting states (and if considering PFAs, all random observations emitted and all random transitions that occur). An *ideal strategy* op^* is a strategy with minimum expected cost.[1]

Learning: The classic control problem involves learning an optimal strategy, given some training information. The "behavioral cloning" [SHKM92, SMB95] approach requires the use of an *omniscient controller*, which is a (possibly stochastic) function $s^* : X \mapsto U$ that maps *states* into actions; this controller follows the optimal action for each state.[2] By analogy to an ideal strategy, an omniscient controller, starting from any starting state x_1, produces a trajectory $\langle x_1, \ldots, x_m \rangle$ where each $x_i \mapsto^{u_i} x_{i+1}$; here, however, this $u_i = ic(x_i)$ depends on the *state* x_i, rather than on the observation of that state $y_i = \ell_{XY}(x_i)$.

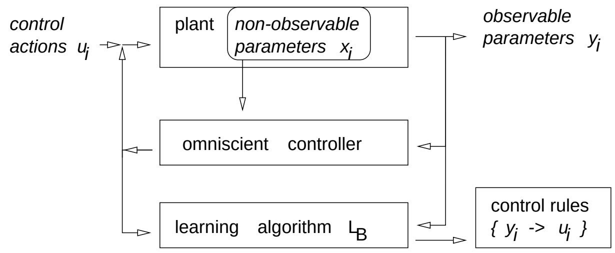

Figure 1: Behavioral Cloning, with Omniscient Controller

The "behavioral cloning" learner L_B watches this omniscient controller as it steers the plant on an ideal trajectory, and so sees information of the form $\langle \langle y_1, u_1 \rangle, \langle y_2, u_2 \rangle, \ldots, \langle y_m, u_m \rangle \rangle$, where each $y_i = \ell_{XY}(x_i)$ is the observation emitted at state x_i, and $u_i = ic(x_i)$ is the appropriate action to take; see Figure 1. The L_B learner's task is to use this information to learn a strategy — *i.e.*, a mapping from observations Y to actions U. *N.b.*, while the omniscient controller knows the actual *states* encountered, the learner will not. Restricting the learner to see only the observables is often reasonable, as the learner's task is to produce a strategy that must decide on appropriate actions based only on such observables. Note also that L_B may see several runs.

By contrast, an "experimental learner" L_E (see Figure 2) does not have access to an omniscient controller. Instead, it will guess some strategy $s_1 : Y \mapsto U$, and test it out, by using it to control the plant. After each run $r^{(j)} = \langle \langle y_1^{(j)}, u_1^{(j)} \rangle, \langle y_2^{(j)}, u_2^{(j)} \rangle, \ldots \rangle$, L_E is told the cost of this trajectory $c(r^{(j)})$.[3] It may then revise

[1] We also consider strategies that take as input a short history of observations; here $op : Y^K \mapsto U$ for some integer K. Other related models consider infinite trajectories, and define the cost as the limit $c(\vec{x}) = \lim_{m \mapsto \infty} \frac{1}{m} \sum_i \ell_c(x_i)$. All of the results in this paper hold if we use a "discounting" term in the sum; $\frac{1}{m} \sum_i \gamma^i \ell_c(x_i)$ for some $0 < \gamma < 1$.

[2] While most work on behavioral cloning assumes a *human* operator is operating the plant optimally, we (along with [MW95, BSW96]) model this ideal operator as a problem solver that determines optimal actions by search; see Section 4.

[3] Note that L_E is not told the cost of each state encountered. Hence, this resembles the related model where actions, rather than states, have costs.

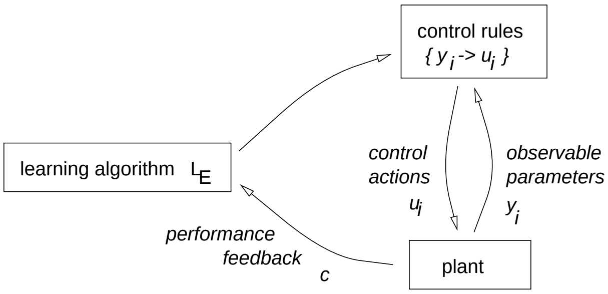

Figure 2: Learning by experimentation

its strategy, to produce a new $s_2 : Y \mapsto U$ strategy, which it can then test out, and further revise if necessary. Note that we allow these strategy modifications to be based on the performance of *all* previously tested strategies — which is very generous, given that typical reinforcement learning algorithms have only an *estimate* of this performance quality, based on that discounted performance feedback that they receive while the hypothesis is modified "on the fly".

In either case, the learner has access to a set of trajectories — L_B has trajectories produced by an omniscient controller (and known to have the optimal score), and L_E has trajectory/score pairs, where L_E produces the trajectories and the "environment" provides the associated score. Our task is to identify when each learner works more effectively; *i.e.*, produces the best controller, using the least input data.

3 COMPARISON

Why L_B May be Better: At first blush, the behavioral cloning learner L_B seems to have the advantage, as it sees precisely the optimal action to take, in an apparently wide variety of situations. By contrast, as the experimental learner has to determine which action is best in a given state, it may have to follow each possible trajectory from a given state to determine the quality of all possible successor states.

To make this point more concrete, imagine a simple "layered" DFA, where one of the k actions $\{u_1, \ldots, u_k\}$, call it $u^{(1)}$, maps the unique starting state $x_1 = x_{init}$ to a useful successor, x_2, while the other $k-1$ actions each map x_1 to the absorbing state x_{bad} — *i.e.*, all k actions map x_{bad} to itself. Exactly one of k actions, call it $u^{(2)}$, maps this x_2 to x_3, while the other $k-1$ actions map x_2 to x_{bad}; and so forth: For each x_i reached, exactly one $u^{(i)}$ maps x_i to x_{i+1}, while the other $k-1$ map it to x_{bad}; see Figure 3. Now assume that all $n+1$ states, $\{x_1, \ldots, x_n, x_{bad}\}$ "look the same"; *i.e.*, $\ell_{XY}(x) = y$, but the costs are very different: $c(x_1) = c(x_2) = \ldots = c(x_n) = 0$, but $c(x_{bad}) = 10000$. Clearly the optimal strategy here is to control this DFA by issuing the commands $u^{(1)}, \ldots, u^{(n)}$, in this order.

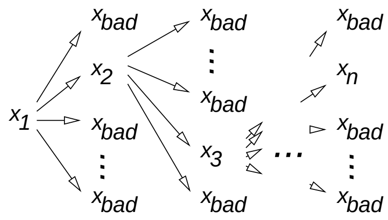

Figure 3: Problem for L_E Learner

The omniscient controller will show the L_B learner exactly this path, which means L_B's induced s_B controller can operate optimally after seeing this single example. By contrast, the L_E learner will have to find this optimal strategy by random chance, which will, in general, require trying out $O(k^n)$ trajectories where there are $O(n)$ states. Hence,

Observation 1 *There are situations where L_B can produce an optimal controller using exponentially fewer samples that L_E.*

Why L_E May be Better: While the L_B learner is given the ideal trajectories, note that this is *all* it is given. This is wonderful if the resulting strategy s_B can always reach, and stay on, such perfect trajectories. Note, however, that s_B is in trouble if it ever departs from this trajectory, as it will then have to deal with a region of the state-space in which it has no experience. Unfortunately, there are several reasons why the s_B strategy will be unable to remain on these perfect paths. First, s_B may not be able to return to an ideal trajectory if an action is stochastic and leads to indistinguishable states; *i.e.*, if $x_0 \mapsto^{u,p_1} x_1$ and $x_1 \mapsto^{u,p_2} x_2$, where $y = \ell_{XY}(x_1) = \ell_{XY}(x_2)$. Here, s^* will know which action to take (as it sees x_1 or x_2), but our s_B cannot, as it only sees y. Similar problems can happen if the observation function is stochastic. Second, even a deterministic automaton may start from different starting states $x_1^{(j)}$; if they all look the same to s_B (*i.e.*, if $\ell_{XY}(x_1^{(A)}) = y = \ell_{XY}(x_1^{(B)})$), then s_B will not know which initial action ($u_A = ic(x_1^{(A)})$ or $u_B = ic(x_1^{(B)})$) to take. In general, we say

Definition 1 An (deterministic) automaton is not *effec-*

tively identifiable if, for some integer $m \in \mathcal{N}$ and possible initial states $x_1^{(A)}$ and $x_1^{(B)}$, the ideal trajectories starting with $x_1^{(A)}$ and $x_1^{(B)}$ — *i.e.*, $r^{(A)} = \langle x_1^{(A)}, x_2^{(A)}, \ldots \rangle$ and $r^{(B)} = \langle x_1^{(B)}, x_2^{(B)}, \ldots \rangle$ — are indistinguishable for the first m steps but require different actions on this m^{th} step — *i.e.*, $\ell_{XY}(x_i^{(A)}) = \ell_{XY}(x_i^{(B)})$ for $i = 1..m$, but $s^*(x_m^{(A)}) \neq s^*(x_m^{(B)})$. ∎

and note that s_B will have problems if the plant is not effectively identifiable.[4]

As an illustration, consider the DFA shown in Figure 4. Here, the controller can see only the first coordinate of each $\langle x, y \rangle$ state, which means $x_{2,1}$, $x_{2,2}$ and $x_{2,3}$ are indistinguishable. However, the ideal trajectory from $x_{2,1}$, $\langle b, a, a, b, a, a, \ldots \rangle$, is significantly different from $x_{2,2}$'s ideal trajectory $\langle a, a, b, a, a, b, a, \ldots \rangle$, etc. If x_{init} could be any of $x_{2,1}$, $x_{2,2}$ or $x_{2,3}$, the L_B learner would only see that the ideal strategy would map the observed "2" to action "a" two times in three, and "b" the rest of the time. The best it could do, here, is map 2 to a (either all of the time, or 2/3 of the time). This, however, is disastrous, as it will lead to the extremely expensive $x_{1,1}$ state.

The L_E learner, on the other hand, would see that performing the b action on seeing "2" is much safer, as it only incurs a loss of t after t time steps, rather than $O(100t)$.

As the inferred s_B controller tries to emulate the omniscient controller, it may enter states (like $x_{2,1}$ or $x_{2,3}$ in Figure 4) with identical observables (here "2"). While the omniscient controller can use its extended perception to always choose a proper action, the cloned controller cannot; this means s_B is likely to enter a disastrous state, which may produce arbitrary worse scores. (Note that L_E was able to avoid this problem by not entering such "risky" states, but instead accepting lower-risk (if lower highest-payoff) trajectories.)

A third problem occurs if s_B has some memory constraints, which prevent it from simply memorizing the ideal sequence of actions. Here, it may have to store instead simple associations, of the form "if see y_i, perform action u_i", perhaps based on frequencies — *e.g.*, if s^* took action u_1 from a state that emitted y 2 times, and action u_2 1 time, than map y to u_1. Unfortunately, this can be catastrophic — *e.g.*, if action u_1 was very bad for this 3^{rd} situation. Notice this is exactly what would happen with the DFA shown in Figure 4, even

[4] Notice a DFA can still be effectively identifiable if there are two or more ideal trajectories that are never distinguished, but always involve the same actions.

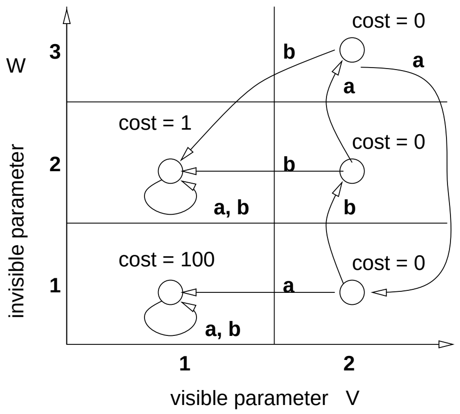

Figure 4: Plant that L_B has difficulty learning

if the $x_{1,1}$ is the only starting state, assuming the controller can only store 2 bits — 1 bit (either a or b) for the "1" state and 1 bit for the "2" state.

In more graphic terms, an omniscient robot could routinely and safely stroll through mine-fields to save time, or run red lights at top speed. The danger to a robot with limited sensors that tried to imitate its omniscient counterpart is obvious.

We can, however, prove that these are all of the caveats:

Theorem 1 *If*

(1) *the plant is a* deterministic *finite automaton,*
(2) *this DFA is* effectively identifiable*, and*
(3) *L_B is able to see, and s_B to record, each complete possible optimal trajectory (i.e., L_B sees the complete trajectory starting from each starting state, and s_B has no memory constraints),*

then the strategy returned by L_B, s_B, will perform optimally (i.e., it will perform as well as s^*). Hence, L_B must be at least as good as L_E in these situations.

Otherwise, if any of the three conditions is violated, there are situations where s_B can perform arbitrarily poorly.

Proof (sketch): ($\Rightarrow$) Here, L_B can trivially give the $\langle\langle y_i^{(j)}, u_i^{(j)} \rangle\rangle_i$ sequences it sees to s_B, which simply "plays back" the u_i sequence that matches the y_is that it observes. Conditions (1) and (2) above guarantee that s_B will eventually be able to identify an appropriate ideal trajectory, and conditions (1) and (3) guarantee that s_B will be able to follow this trajectory.

($\Leftarrow$) The comments above sketch the proof of this direction; the extended technical report [SGD97] provides more details. ∎

As an immediate corollary, note that L_B is a good learner if (it has arbitrary memory and) there is no perceptual aliasing; *i.e.*, if $\ell_{XY}(x) = \ell_{XY}(x') \Leftrightarrow x = x'$. Note, however, that s_B may be sub-optimal even if there is no perceptual aliasing.

Given conditions (1)–(3), we can use the above proof to bound on the number of trajectories that L_B needs to observe: Since $s^* : X \to U$ is a well-defined function, there is only one trajectory starting from each initial state. L_B will therefore gather all of information it needs after seeing one trajectory associated with each of the $|x_{init}| \leq |X|$ possible starting states. With high probability, L_B will see at least one instance of each of the "not-unlikely" runs if it randomly observes $O(n \ln n)$ trajectories, where $n = |X|$ is the total number of states.[5] By contrast, we saw above that L_E can require $O(k^n)$ trajectories to learn a $O(n)$-state DFA.

Of course, issues like perceptual aliasing are also problematic for the L_E experimental learner. That learner, however, will have some experience with other parts of the search space, besides the straight-and-narrow line provided by s^*. In fact,

Theorem 2 *Given arbitrarily large number of samples (covering all possible runs), L_E will, with probability* 1*, obtain the information required to perform optimally on a DFA.*

Proof (sketch): Eventually L_E is (wp1) guaranteed to see every trajectory that can be encountered, from every possible initial state, enough times to obtain the statistics required to determine the appropriate action to take from each situation. ∎

Two comments: First, even given arbitrarily many samples, L_B may not converge to an optimal learner, if either condition (1) or condition (2) of Theorem 1 fail to hold. Second, even given this information, L_B is still left with the challenge of computing which action is appropriate in each situation (*i.e.*, given a history of observable/action pairs). If s_B has any memory limitation, it will eventually (for large enough P/DFA) have to find a non-trivial representation, shorter than simply listing the set of complete trajectories. This challenge — of producing a small DFA consistent with a set of observations — is intractable, and in fact, is not even approximatable [PW93].

3.1 DISCUSSION AND RELATED WORK

Note that the different learners — L_B and L_E — are receiving very different information: L_B does get perfect information (*i.e.*, the optimal action to take) but only about a very narrow of situations; by contrast, L_E can obtain information about wide range of situations, but each sample is only a "noisy" estimate of the desired observation-to-action mapping.

Behavior cloning has an obvious connection to the standard machine learning work on classification [BMSJ78]: In each case, the learner explicitly sees the optimal answer for each given situation. Here, however, both the training and the test samples are missing some information — *i.e.*, the state. Moreover, these samples are not iid (independent and identically distributed), as the optimal u action depends not just on the observed y, but also on the unobserved state x information, which can depend on earlier states in the trajectory.

Here, this is further confounded by the problem that the learned controller s_B performance system may later be tested on states that the learner L_B never encountered, as s_B cannot, in general, be expected to stay on the ideal trajectory.

This argues that a good learning environment should give the learner access to a wide range of situations, to allow (force?) the L_B-ish learner to explore other regions of the space. Several researchers have provided such environments: For example, Pomerleau's NavLab system [Pom93] learned to drive by watching a human driver's steer, in response to the visual observations of the current road. Unfortunately for NavLab (but fortunately for these drivers), people tended to drive effectively — which meant NavLab had little experience *not* being in the appropriate lane position. Pomerleau therefore presented NavLab with a variety of artificially produced scenes (formed by distorted some real scenes), each coupled with the proper steering-action. Similar techniques have been used to produce effective Face Recognition systems, that can work robustly over a wide range of conditions [SP93]. As a third example of this phenomenon,

[5] Technically, after observing $\frac{|x_{init}|}{\epsilon} \ln \frac{|x_{init}|}{\delta}$ trajectories whose starting states are drawn at random from the "x_{init} distribution", L_B will, with probability at least $1-\delta$, have seen at least one run of every optimal trajectories that occurs with probability at least $\epsilon/|x_{init}|$. This means (with high probability) the induced s_B will be perform optimally at least $1-\epsilon$ of the time.

Epstein also realized that her game-playing program would not work robustly against a range of opponents unless it was trained against a similarly large range of challengers [Eps94].

Finally, our model has several significant differences from the standard (adaptive) linear control framework [NA89], which usually assumes the learner knows a lot about the plant's basic model in advance (*e.g.*, that the plant is linear, and perhaps order bounds on the numerator and denominator of the transfer function that represents the system), and tends to evaluate learners on-line, in terms of a loss function. In our framework, by contrast, our learners do not have a model of the plant in advance, and they are evaluated based on the quality of induced controller, after the completion of the learning phase. While there are mappings that connect these models, such analyses are beyond the scope of this paper.

4 EMPIRICAL RESULTS

Here, we consider the task of controlling an aeration tank used in a wastewater plant. With more oxygen, the ammonium contained in the water is converted to nitrate. Moreover, any oxygen remaining in the tank will reduce the efficiency of the anaerobic bacteria, which are intended, in a second step, to convert the nitrate to free nitrogen, which then leaves the system. Our task is to keep the emission of ammonium on a steady low level, while minimizing both the consumption of oxygen and emission of nitrate. The aeration rate may be set to any of six possible values. For our experiments, we used a simulator that provides a very precise model of the process by numerically integrating the underlying differential equations. The state of the process is determined by 104 parameters representing concentrations of various chemical and biological substances in the reaction tanks; however, we only measured two of them, forcing us to deal with strong perceptual aliasing. Since the amount and pollution of water flowing into the plant is not predictable based on our measurements, state transitions are also non-deterministic. The quality criterion we are optimizing is the sum of two components: a very strong punishment for exceeding a tolerable ammonium concentration at any time (which may damage the eco-system), and a punishment proportional to the consumption of oxygen.

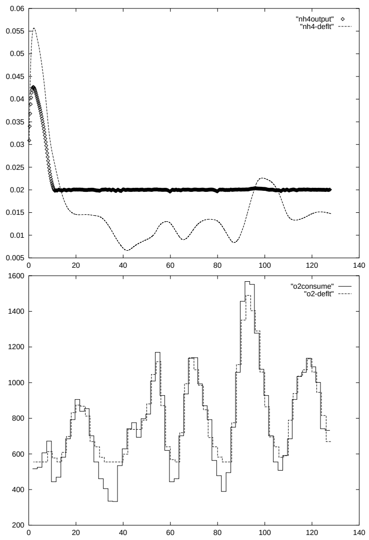

Figure 5: Ammonium (top), oxygen consumption (bottom) of optimal controller (bold), compared to those of the hand-crafted controller (thin lines)

4.1 OMNISCIENT CONTROLLER

We generated a number of optimal sample trajectories using the problem solving approach: as the full effect of a control action can only be measured four hours after the action was performed, to decide on an action for a given state we exhaustively search the tree of possible actions to a depth of four hours, then choose the best action, and continued this procedure from the resulting state. This procedure yields the optimal trajectory from a given starting point. Note that this is only possible as this omniscient controller has complete knowledge of both the non-observable state parameters and the future input of wastewater to the plant.

Figure 5 shows the ammonium and oxygen curves of this controller (bold line) over a 130 hour period, compared to the controller currently installed (thin line). We can use our knowledge of the optimal control actions for some states to quantify the perceptual aliasing we are facing: we can determine the information gain, *i.e.*, the amount of information given by the parameters, on which action to choose in a given state. The information gain of the two observable parameters is 0.05 and 0.04 bit per test respectively, while some of the non-observable parameters would provide up to 0.3 bits per test. In order to tell which of six possible control actions is optimal, we need $\ln_2(6) = 2.58$ bits of information (all actions are about equally likely). Given that the observable parameters provide only 0.05 bits each, it is very unlikely that any combination of them will provide enough information to identify states. Considering the measurements of the past 4 hours, we are able to gain up to 0.055 bits. Knowledge about the past measurements increases the information about the optimal action, but the fact that the maximum does not exceed 0.055 bits provides strong evidence that the system is not effectively identifiable.

4.2 LEARNING A BEHAVIORAL CLONE

Figure 6 shows the optimal control action (y-axis) over the value of one of the observable parameters (x-axis). Clearly, there is no obvious correlation between these values. We used an L_B-style algorithm for induction of ripple down rules (*i.e.*, rules with nested exceptions [Sch96, Sch95]) to describe the relation between this observable parameter and the optimal action via rules. We learned rules with an accuracy of up to 25%, but the rules were completely unable to keep the plant in a stable state. We then determined a cost matrix for sub-optimal actions, whose i, j element measures the long-term cost of choosing operation i when operation j would be optimal. We found that one action caused extremely high penalties when not performed in a situation where it would be optimal. The rules minimizing the expected misclassification costs therefore contained the single rule: always predicting that action. Of course, this rule does not effectively control the plant!

A controller learned by the DIPOL classification system [SW94] — a hybrid statistical/neural algorithm, that was ranked the best learning algorithm, on average, on the European StatLog project [MST94] — kept the system in a stable state, but the controller consumed an extremely large amount of oxygen. Providing the second observable parameter and the past measurements of both observable parameters as input reduced the oxygen consumption slightly (1-2%) for both algorithms, but the results were still unacceptable. Note, that the hand-crafted controller, which also sees only one parameter, performs better than the controllers generated by the learning algorithms. It is, however, unfair to blame the learning algorithms for this bad performance; Applied to the sample states, the hand-crafted controller yields the optimal action less than 15% of the time, while the rules achieve an accuracy of between 25% and 30%.

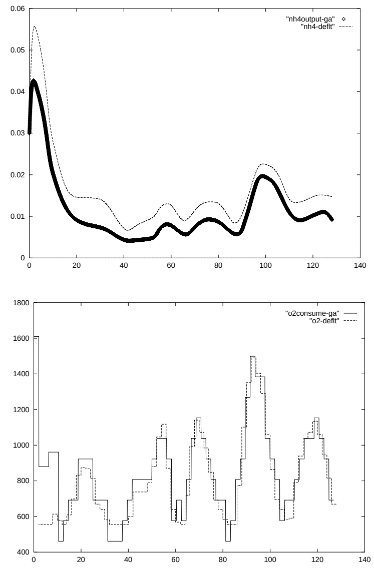

Figure 7: A controller using only the observable parameters

4.3 EXPERIMENTAL LEARNING

As our hypothesis language, we chose a list of rules with interval constraints for the present and past measurements of observable parameters, with exactly one rule per possible action, where the ordering of the rules is significant. Our L_E-style learning algorithm proposes a controller, which then controls the plant for 130 hours of simulated time. The overall costs are returned to the learning algorithm, which then performs a gradient descent search for a good controller. We modified the controllers by adding normally distributed random numbers to all interval boundaries, and further used a simple mechanism for step-size adaptation, as used in evolutionary strategies [Rec94, Rec89], based on comparing two successors generated with different step sizes. After about 10 minutes of computation time, we found a fairly good controller, which takes only one observable parameter as input. Figure 7 shows its ammonium (left) and oxygen (right) curves. Note it is better than the hand-crafted controller but worse than, and completely different from, the omniscient controller.

4.4 EXPERIMENTAL RESULTS

The dotted line in Figure 6 refers to the control strategy found by the experimental learner; notice this control action is (roughly) inversely proportional to the

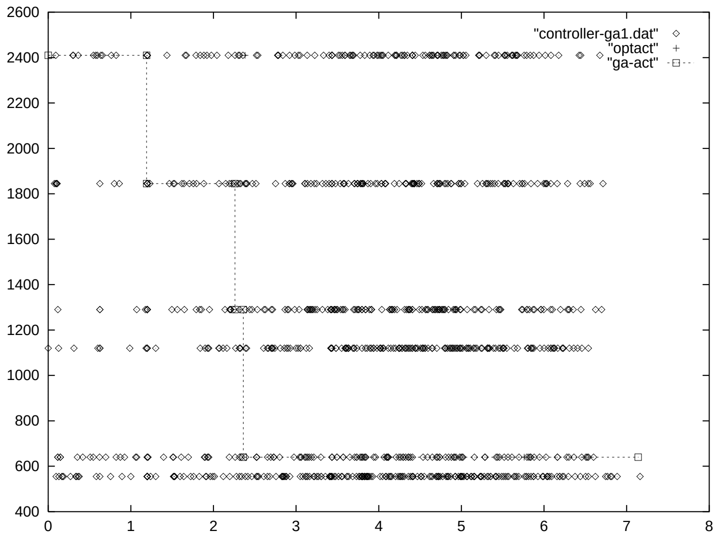

Figure 6: Observable parameter (x-axis) and optimal action (y-axis) found by problem solver. Dotted line shows strategy found by experimental learner

value of the observable parameter. Comparing this strategy to the optimal control actions determined by the problem solver, we see that the experimental controller performs the optimal action for only 15% of the sample states, while the extracted rules perform the optimal action for 25% of the sample states and yet perform significantly worse! That is, *although the rules approximate the projection of the optimal strategy to the space of visual parameters better than the experimentally obtained controller does, the experimental controller achieves the better result.* This is because the mimicking controller is unable to simulate the optimal controller with unlimited perception, but the experimental learner was able to find a strategy that yields good results using only one observable parameter — albeit a strategy that is very different from the optimal one.

This clearly shows that the effects of Theorem 1 can be observed in real-world systems, even though our system is continuous while we only proved that Theorem 1 applies to discrete automata.

5 CONCLUSION

This paper presents a variety of results that identify some of the strengths, and weaknesses, of two methods for learning control information. We first consider learning from arbitrarily many samples, and prove that an experimental learner L_E will receive the information required to learn an optimal controller, even if there is perceptual aliasing. By contrast, we show that a behavioral cloning learner L_B will only learn an optimal controller if the underlying system is effectively identifiable, and may perform arbitrarily poorly if there is perceptual aliasing. We next consider the "limited sample framework", and ask how many samples each of these learners requires to produce its best controller. Here we prove that L_B requires relatively few samples (essentially linear in the number of DFA states) to produce its best controller (bad as that controller may be). However, L_E may require a number of "experiments" that grows exponentially with the number of states on the optimal trajectory.

Our experiments provide strong evidence that these worst-case results apply to real-world plants. Although the behavioral clone s_B did approximate the omniscient controller better than the experimentally learned s_E, this s_E performed reasonably while s_B did not, as s_B was unable to keep the plant in a steady state, or kept it in an unacceptably bad one.

To conclude: Our results show that behavioral cloning can be more efficient than experimental learning, especially if we do not have to deal with perceptual aliasing. However, cloning is potentially less robust. In particular, it may perform arbitrarily poorly (and hence much worse than the experimental learner) in the presence of perceptual aliasing.

Acknowledgment

This work was partially supported by an Ernst-von-Siemens fellowship held by Tobias Scheffer.

References

[BMSJ78] Bruce G. Buchanan, Thomas M. Mitchell, Reid G. Smith, and C. R. Johnson, Jr. Models of learning systems. In *Encyclopedia of Computer Science and Technology*, volume 11. Dekker, 1978.

[Bro93] Martin Brooks. Proposal for a pattern matching task controller for sensor-based coordination of robot motions. In *Robots and Biological Systems*. Springer-Verlag NATO ASI series F, 1993.

[BSA83] A. B. Barto, R. S. Sutton, and C. W. Anderson. Neuronlike adaptive elements that can solve difficult learning control problems. In *IEEE Transactions on Systems, Man, and Cybernetics*, 1983.

[BSW96] L. Briesemeister, T. Scheffer, and F. Wysotzki. A concept-formation oriented approach to skill acquisition. In *Proc. European Workshop on Cognitive Modelling*, 1996.

[DB95] Richard C. Dorf and Robert H. Bishop. *Modern Control Systems*. Addison-Wesley, Reading, Mass., 7th edition, 1995.

[Eps94] Susan L. Epstein. Toward an ideal trainer. *Machine Learning*, 15:251–277, 1994.

[Kae93] Leslie Pack Kaelbling. *Learning in Embedded Systems*. MIT Press, 1993.

[Kha96] Roni Khardon. Learning to act. In *AAAI96*, Portland, August 1996.

[MC68] D. Michie and A. Chambers. Boxes: An experiment in adaptive control. In *Machine Intelligence 2*, 1968.

[MST94] D. Michie, D. J. Spiegelhalter, and C. C. Taylor. *Machine Learning, Neural and Statistical Classification*. Ellis Horwood, 1994.

[MW95] Wolfgang Müller and Fritz Wysotzki. Automatic synthesis of control programs by combination of learning and problem solving methods (extended abstract). In *Machine Learning: ECML-95*, pages 323 – 326, 1995.

[NA89] K. Narendra and A. Annaswamy. *Stable Adaptive Systems*. Prentice Hall, 1989.

[Pom93] Dean A. Pomerleau. *Neural Network Perception for Mobile Robot Guidance*. Kluwer Academic Publishers, 1993.

[PW93] L. Pitt and M. Warmuth. The minimum consistent DFA problem cannot be approximated within any polynomial. *J. ACM*, 40(1):95–142, 1993.

[Rec94] I. Rechenberg. *Evolutionsstrategie '94*. Frommann-Holzboog, 1994.

[Rec89] I. Rechenberg. Artificial evolution and artificial intelligence. In R. Forsyth, editor, *Machine Learning*, pages 83–103, London, 89. Chapman.

[RN95] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 1995.

[SCG91] A. Carlisle Scott, Jan E. Clayton, and Elizabeth L. Gibson. *A Practical guide to knowledge acquisition*. Addison-Wesley Pub Co., Reading, MA, 1991.

[Sch95] T. Scheffer. Learning rules with nested exceptions. In *Proc. International Workshop on Artificial Intelligence Techniques*, Brno, Czech Republic, 1995.

[Sch96] T. Scheffer. Algebraic foundation and improved methods of induction of ripple down rules. In *Proc. Pacific Knowledge Acquisition Workshop*, 1996.

[SGD97] Tobias Scheffer, Russell Greiner, and Christian Darken. Why experimentation can be better than "perfect guidance". Technical report, Siemens Corporate Research, 1997.

[SHKM92] C. Sammut, S. Hurst, D. Kedzier, and D. Michie. Learning to fly. In *ICML92*, Aberdeen, 1992.

[SMB95] B. Schulmeister, W. Müller, and M. Bleich. Modelling the expert's control behavior by machine learning algorithms. In *Proc. International Workshop on Artificial Intelligence Techniques*, 1995.

[SP93] Kah-Kay Sung and Tomaso Poggio. Example-based learning for view-based human face detection. Technical report, Massachusetts Institute of Technology, 1993.

[SW94] B. Schulmeister and F. Wysotzki. The piecewise linear classifier DIPOL92. In F. Bergadano and L. De Raedt, editors, *Machine Learning: ECML-94*, LNAI 784. Springer Verlag, 1994.

[WB91] S. D. Whitehead and D. H. Ballard. Learning to perceive and act. *Machine Learning*, 7:45–83, 1991.