Modified Reinforcement Learning for Sequential Action Behaviors

and its Application to Robotics

Chris Robinson
Dept. of Electrical and Computer Engineering
University of Louisville Speed School of Engineering
Louisville, KY

Abstract— **When developing a robot or other automaton, the efficacy of the agent is highly dependent on the performance of the behaviors which underpin the control system. Especially in the case of agents which must act in real world or disorganized environments, the design of robust behaviors can be both difficult and time consuming, and often requires the use of sensitive tuning. In response to this need, we present a behavioral, goal-oriented, reinforcement-based machine learning strategy which is flexible, simple to implement, and designed for application in real-world environments, but with the capability of software-based training. In this paper, we will explain our design paradigms, the formal implementation thereof, and the algorithm proper. We will show that the algorithm is able to emulate standard reinforcement learning within comparable training time, and to extend the capabilities thereof as well. We also demonstrate extension of learning beyond the scope of training examples, and present an example of a physical robot which learns a sequential action behavior by experimentation.**

Keywords—Machine learning; Robot control; Robots; Reinforcement learning; Algorithms; Probabilistic learning; Behaviors; Behavior Based Robotics; Operant Conditioning;

I. Introduction

Many robotic paradigms are inspired by biological exemplars- for example the ever popular 'robotic insects', which are designed under the reactive paradigm to have simple relations which lead to comparatively complex behavior, especially when compared to the effective information processing capacity of these robots. In that same vein, we approach our task by examination of the principles of operant conditioning as a psychological model of the type of behavior we seek in our agents.

To this end, we propose a restructured reinforcement learning algorithm designed to incorporate elements inspired by examination of operant conditioning learning ([1]). Additionally, we propose a modified interpretation of the reinforcement learning paradigm which enables us to more effectively utilize precepts from operant conditioning research to achieve our functional goals. We also establish a simple mechanism of creating behavior sequences which arise organically from architecture, rather than from designer specified hierarchies, a training algorithm based on the procedure of successive approximations, and examine several experimental results to establish feasibility, efficacy, and functional advances as compared to standard reinforcement learning.

More specifically, we will show that the modified architecture is performance comparable to Q-Learning, and that the modification is an effective learning algorithm on a standard problem. We will also demonstrate functionality in a context in which the unmodified reinforcement learning is unable to perform effectively, show that the system is capable of functioning in a non-idealized version of the training environment, and that it is resistant to noise in the training signal. These last two aspects are highly pertinent to robotics applications.

Our approach is characteristically related to other reinforcement learning methods, although our design paradigm leads to creation of these certain similarities in very different ways. We would first like to note the relation to Hierarchical Learning such as [2]. However, our algorithm takes a bottom-up approach to behavior generation, rather than a top-down decomposition method.

We would also like to note [3] and [4], which bear striking philosophical similarities to our work. First, with regards to [3], we note that although there is a similarity in paradigm in the attempt to create behavior sequences in the form of the subsumption architecture, out algorithm incorporates recurrence to cause linking and chaining of behaviors, negating the need for specification of any subsumption model. With regards to [4], we remark that the development of chains occurs within our architecture by correlation of inputs to actions, as opposed to the creation of explicit relations via predication ([4]) or explicit linking ([6]).

Finally, we make the general statement that, unlike most machine learning schemes which implement sequential behaviors, our algorithm copes with the problem of temporal credit assignment by applying reinforcement across temporally correlated actions, as opposed to statistical inference among states. Additionally, our architecture does not make use of explicit releasers, as in [3] and [5]. This is possible in our case because linking occurs due to recurrence, rather than correlation identification.

II. ALGORITHM

As mentioned before, there are several architectural modifications that comprise our modification of reinforcement learning. In particular, we will be implementing a property we refer to as temporal linking. We will also consider an alternative interpretation of reinforcement learning which is inspired by psychology, specifically that an agent can be viewed as a probabilistic decision machine, as opposed to an estimator of quality of state.

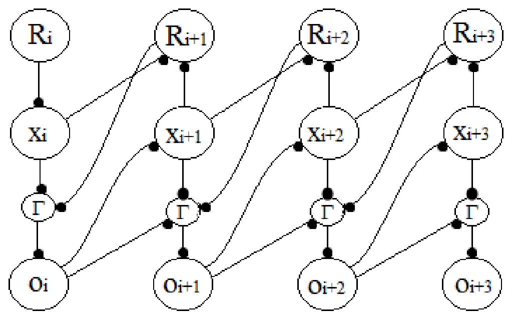

Fig. 1. Process model; $\mathbf{x}_i$ are sense-observable states, $\mathbf{o}_i$ are action outputs, R_i are reinforcement signals and Γ represents the action of the reinforcement learning system. For clarity, influence lines for the historical set update are not shown here.

Temporal linking is the most important contribution from the psychological analogy. In operant conditioning, there is a link between each behavior in a sequence, which indicates the next action which leads to a reward. Without this dynamic aspect, sequential behaviors are beyond the reach of the agent. The means by which we establish this dynamism is twofold: rewarding historical sequences of state action pairs as a set, encouraging extraction of statistical patterns in the temporal domain, and by the establishment of a loop relation between successive actions taken by the agent.

We take the direct route of concatenating the most recent action with the sense input of the next decision. In this way, the current step is a function of the state as observed by the agent, and of the previous actions. We will represent this link by an augmented state vector in which the first components represent the current observations, and the remainder represent the prior output (and therefor the history of the agent, albeit in an abstract fashion). Figure 1 shows the process model, illustrating the relationship between sense data, prior actions, and reinforcement. This diagram does not represent the historical set update rule, which we discuss below, for aesthetic reasons.

The second element of temporal linking is the reinforcement of historical state action pairs, so that the agent is capable of extracting patterns over multiple actions. That is to say, when we apply reinforcement to the agent, we apply the same signal to some number of the previous SA pairs, kept by the agent in a queue. We apply the change based on the current reward (possibly at a diminishing rate) to successively older states. This mechanism of set updates is a means to allow a sequence of actions to become statistically linked based on the utility of the sequence.

Finally, we have the modified interpretation of reinforcement learning (note that our algorithm is built around some preexisting reinforcement structure, such as Q Learning). Typically, one considers the output from a reinforcement learning agent to represent the predicted quality of a decision. We consider instead a similar but subtly different interpretation- that the output value represents the probability that taking the specific action in the given state will lead to a positive reinforcement signal. This interpretation recasts the values in terms suitable for a weighted stochastic decision maker. We implement this via action selection policy- the outputs of the RL subsystem for each action, given the current observational state are tabulated, and a cumulative distribution is constructed. A weighted random selection is then made from within this distribution.

To close this section, we present a flow diagram of the system in Figure 2, illustrating the process flows in terms of inputs to various sub modules.

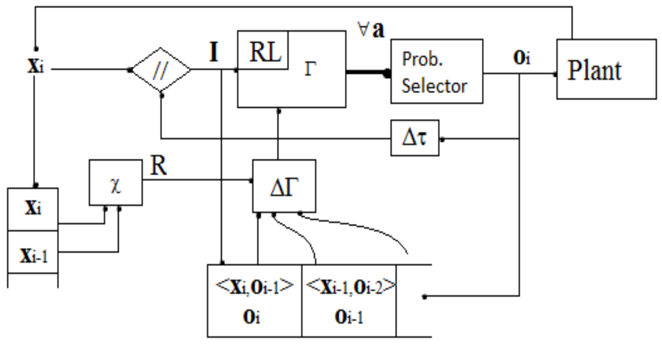

Fig. 2. Flow diagram showing the full architecture- RL subsystem, probabilistic output selector, critic χ, and historical queue. Note that as the reinforcement system will vary, Γ and ΔΓ are general.

III. FORMALIZATION

From a mathematical standpoint this systemic modification is relatively simple, for the reinforcement module (we use **I** to denote generalized input):

$$\Gamma(\mathbf{I}_t) = \mathbf{o}_t \qquad (1)$$

$$\Delta\Gamma_t(\mathbf{o}_{t-1} \mid \mathbf{I}_{t-1}) = (\alpha)\delta[r(t),\Gamma(\mathbf{o}_{t-1},\mathbf{I}_{t-1})] \qquad (2)$$

Where Γ represents the probabilistic mapping of an input vector **x** onto an output **o** (which can be either a low-level output or an elementary action), and $\Delta\Gamma_\tau(\mathbf{o}|\mathbf{I})$ represents the modification to a specific $(\mathbf{I}_{t-1},\mathbf{o}_{t-1})$ mapping pair due to a reinforcement signal r(t) applied at time t. Additionally, let us make the relation:

$$P(r(\tau)=1 \mid \mathbf{o}_\tau=\mathbf{o}_i, \mathbf{I}_t) = P[r(t) = 1 \mid \mathbf{o}_\tau=\mathbf{o}_i, \mathbf{I}_t] + \alpha\delta[r(t),\Gamma(\mathbf{o}_t|\mathbf{I}_t)] \qquad (3)$$

Which is to denote the interpretation of Γ as representing the probability of positive reinforcement at time τ given that the agent takes action $\mathbf{o}_i$ from observable state $\mathbf{x}_i$, where τ is the next time when the input is **I**t. This relation represents the probabilistic interpretation. To choose the next action, we construct the cumulative distribution of the outputs for all actions given the current state $\mathbf{I}_i$:

$$O_{cdf} = \{\Gamma(\mathbf{o}_1|\mathbf{I}_i), \Gamma(\mathbf{o}_1|\mathbf{I}_i)+\Gamma(\mathbf{o}_2|\mathbf{I}_i), ..., \sum[\Gamma(\mathbf{o}_j|\mathbf{I}_i)]\} \quad (4)$$

we then generate a random number, R_{sel}, from a uniform distribution. Our selection is then the index of the minimum element in O_{cdf} that is greater than or equal to R_{sel}:

$$\mathbf{o}_{i+1} = \min(\{k \mid R_{sel} \leq O_{cdf}[k]\}) \quad (5)$$

This constitutes the probabilistic decision engine. Next, we have the temporal linking property. The first portion thereof we represent thus:

$$\Gamma_t(<\mathbf{x}_t,\mathbf{o}_{t-1}>) = \mathbf{o}_t \quad (6)$$

where $<\mathbf{x}_t,\mathbf{o}_{t-1}>$ is the vector concatenation of the sensor data vector at the current step ($\mathbf{x}_t$) and the action taken at the previous time step ($\mathbf{o}_{t-1}$).

The second component of temporal linking is the historical set update procedure. We introduce a new system parameter which we refer to as memory depth, denoted m_d. This parameter specifies the length of the queue in which the historical information is retained (the augmented vector, along with the concurrent action). We represent this queue by:

$$H = \{(<\mathbf{x}_t,\mathbf{o}_{t-1}>,\mathbf{o}_t),(<\mathbf{x}_{t-1},\mathbf{o}_{t-2}>,\mathbf{o}_{t-1}),... (<\mathbf{x}_{t-md+1},\mathbf{o}_{t-md}>,\mathbf{o}_{t-md+1})\} \quad (7)$$

At each training step, we apply r(t) to each state-action pair: $(<\mathbf{x}_k,\mathbf{o}_{k-1}>,\mathbf{o}_k)$, in H (which includes the most recent pair). Generally we will apply a geometrically decreasing coefficient to the older SA pairs, assuming that older pairs are less likely to have contributed to the most recent reward. Letting the discount for the kth element of H be γ_k, we have a set of updates whose form will vary depending on the RL system, but which will have an effect which can be expressed in terms of the probabilistic interpretation as:

$$\Gamma'(\mathbf{o}_{t-1}|\mathbf{x}_t) = \Gamma(\mathbf{o}_{t-1}|\mathbf{x}_t) + \alpha(\gamma_1)\,\delta[r(t),\Gamma(\mathbf{o}_{t-1}|\mathbf{x}_t)] \quad (12)$$
$$\Gamma'(\mathbf{o}_{t-2}|\mathbf{x}_{t-1}) = \Gamma(\mathbf{o}_{t-2}|\mathbf{x}_{t-1}) + \alpha(\gamma_2)\,\delta[r(t),\Gamma(\mathbf{o}_{t-2}|\mathbf{x}_{t-1})]$$
$$...$$
$$\Gamma'(\mathbf{o}_{t-md+1}|\mathbf{x}_{t-md}) = \Gamma(\mathbf{o}_{t-md+1}|\mathbf{x}_{t-md}) + \alpha(\gamma_{md})\,\delta[r(t),\Gamma(\mathbf{o}_{t-md+1}|\mathbf{x}_{t-md})]$$

Summarily, an r(t) signal of 0 causes the corresponding probability for a state action pair to tend to zero, and a signal of 1 causes a tendency towards an occurrence probability of 1. This allows the agent to develop statistical patterns by interaction with the first temporal linking property.

IV. Experiments and results

In this section, we discuss five different experiment batteries performed using this architecture. The second, third, and fourth of these are simulations. The first experiment is also computerized, however it pertains to a non-dynamical learning task, and is not truly simulation proper. The fifth experiment is a physical implementation of the architecture in a mobile robot, for qualitative examination of implementation efficacy. Throughout these experiments, we apply the architecture to a Q Learning reinforcement system; we refer to this modified system as STQL.

A note on experimental technique needs to be made in relation to the use of a 'random' agent. For the purposes of analysis of learning, it is useful to compare an agent to another, stochastic problem solver, providing a useful baseline comparison to establish that the agent is learning. The random agent is, for purposes of quality of comparison, sanitized. By this we mean that the random agent does not take senseless actions, such as rotating left, then right (which costs two cycles without creating any net change in state) or driving into a wall (a no-op).

The first experiment is principally for establishment of fitness of the architecture as a learning algorithm. In this case we consider the standard XOR learning problem. Herein, we actually consider two related tests- the standard single XOR (XOR), and a paired complement variation thereof (2XOR). The paired complement version is a two-output system, where the first output is congruent with the XOR operation and the other with its complement. By comparing the XOR performance curves to the 2XOR curves, we can make some deductions about information handling with scaling of problem complexity.

Figures 3 and 4 show the performance curves for the XOR and 2XOR problems for both QL and STQL learning, plotted in terms of number of correct classifications. First, we note the efficacy of STQL in the XOR context, in which STQL learns the pattern at approximately 53% of the speed of QL. This does not at face value seem to be particularly strong performance, except that STQL has twice as many states to learn as QL, due to the augmented state input. Although it would be naive to assume a linear relationship between learning speed and number of states, it is still intuitively obvious that these figures indicate at least reasonable performance of the STQL agent as compared to the QL agent.

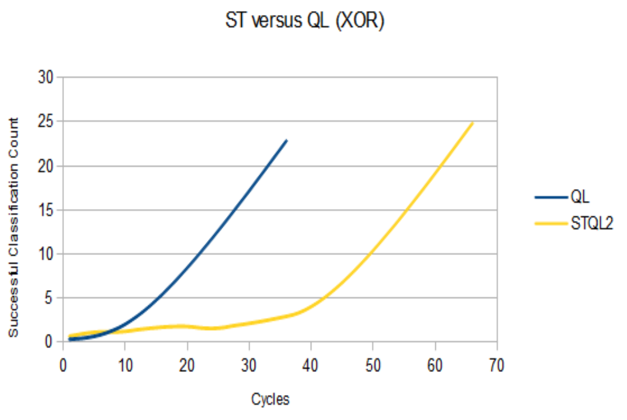

Fig. 3. Performance Curves for QL and ST on the XOR problem. Note that STQL is reasonably slower (learning at approximately 53% the speed of QL), as it has roughly twice as much information to process at each training step, however it is still comparable to QL.

Where the true point of interest is, however, is the comparison of Figure 3 with Figure 4. In the latter, we have the comparison of learning curves for the 2XOR problem. In this case, STQL learns at about 74% of the rate of QL. There is an observation that makes this particularly telling: we note that there are now four 'actions' (2XOR has 2 binary outputs) that an agent can take. As a result, STQL has four times as many states as QL (and twice as many as for XOR) and yet shows improvement in comparative speed over the XOR case.

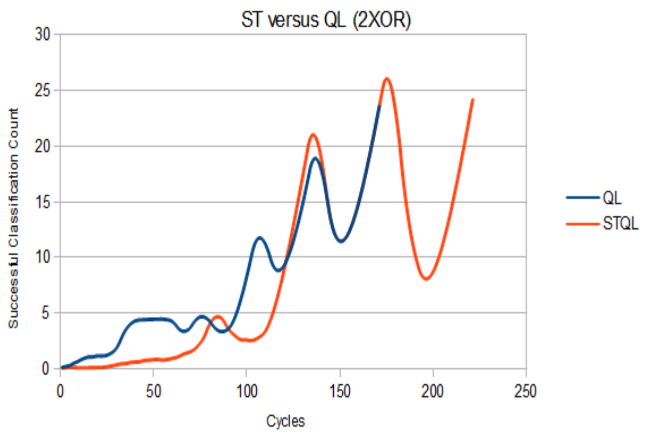

Fig. 4. Learning Curves for QL and ST on the 2XOR problem. Note that in the scaled complexity problem, QL is still somewhat faster, however the STQL is not linearly scaled in speed versus complexity, despite still having twice the information capacity as QL, STQL performs at roughly 74% of the speed of QL- indicating improved information processing vis-a-vis comparison to the XOR example.

Our second experiment begins to take us in a more practical direction. We investigate the performance of the QL and STQL agents on a taxi problem, in which the agent is required to navigate to the location of a 'passenger', perform a pickup operation, navigate to a 'destination', and then execute a 'drop-off' operation. In the construction of the agent for this problem, we apply the sense input in the form of the relative position to the goal- in terms of greater-than, less-than, or equal-to. The action set consists of cardinal direction movements- north, south, east or west, and pick-up and drop-off actions (separately). Goal and initial locations are selected randomly from within a 10x10 gridded world, excepting cases where the goal locations or start location would coincide with one another. Reinforcement feedback is provided by reinforcing actions which reduce the city-block distance from agent to goal, with the inclusion of negative reinforcement for a drop-off or pick-up performed at the wrong location.

The first portion of this experiment is examination of training patterns for STQL and QL. Figure 5 illustrates two average performance curves for each, showing very close to parallel average performance. Performance is compared here as an average principally because individual curves are highly influenced by the starting conditions. By examination, we can easily see the qualitative similarity in the learning rates of the two agents, which shows remarkable improvement over the comparisons noted in the simpler XOR case.

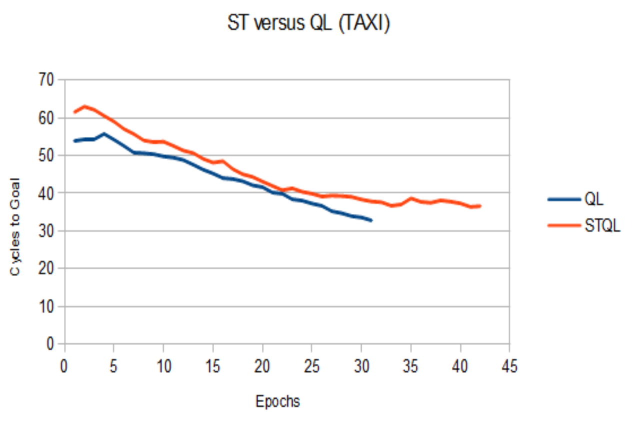

Fig. 5. Learning curves for STQL and QL on the taxi problem, showing close parallel convergence. In contrast to the XOR examples, the difference between the two is minor, with statistical variance of early samples playing a major role in the net speed of training. Note that these are parameter-tuned instances, and that the curves are averages over several training instances.

Figure 6 illustrates an example learning curve for STQL. From the unsmoothed data, it is possible to see the effect of the random alignment of goal states. This learning curve is displays exponential decay of length of epoch, measured in terms of cycles necessary to reach the goal state. For situations in which an epoch is well defined (i.e. when goal achievement is a non-instantaneous procedure), we observe this exponential structure as the most common learning curve. Of particular note is that in the majority of cases (that is, cases in which the first sample is a good representation of the problem in general) the majority of the agent's learning occurs during the very first epoch, evidenced by the extreme precipitous increase in performance thereafter. Although one can see learning occurring in later epochs, the scale of the change in performance indicates that this learning is more akin to tuning than insight.

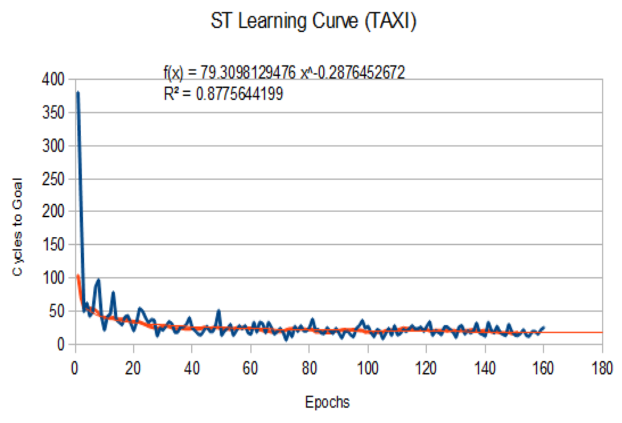

Fig. 6. An example of a learning curve for STQL on the taxi problem. Although the exact bounds of the curve depend highly on the distribution of early training samples, this shape trend is common over large numbers of trials, and the STQL learning curve is, in general, exponential.

The principle goal of this second battery of experiments was investigative. That is to say, we do not seek to illustrate any particular utility of STQL, but rather to establish certain parameters of operation- first and foremost demonstration that the agent does solve problems. Second, it is critical for real world applications that training occur within a realistic time frame. We establish viability in this arena via comparison to QL, wherein we see explicit close learning rate matching. Additionally, we use this problem to explore the general trends in learning behavior, identifying a trend of exponential learning curves.

The third battery of experiments is the perhaps the most qualitatively interesting, and helps to establish the utility of the properties of the STQL agent beyond standard Q learning. In this experiment, we apply the agents to a maze problem. The problem domain is (initially, we will modify this later) comprised of a [12x12] simply 4-connected maze with uniform pathway widths. The agent's starting position and goal position are randomly selected. Agents collect sense data as wall observations from the locally oriented left, right, and ahead directions. The agent's actions are to turn clockwise or counter-clockwise, or to move forward. The critic assigns reward based on a wall-following agent.

Our first experiment is examination of the performance of the QL agent in this problem space. Figure 7 presents learning curves for both QL and the random agent in the maze. Recall that although QL is often used to learn maze problems, that the problem phrasing is quite different from the standard in this case- goal locations and starting locations are random, and states are local- not locations within the maze but sense information observable from the agent's current location.

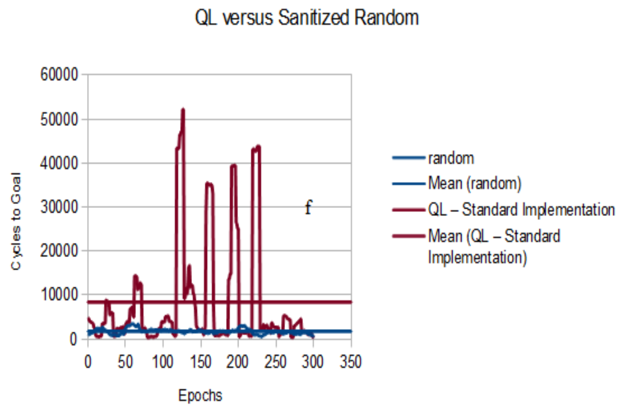

Fig. 7. Graphical demonstration of the application of QL to the maze problem, showing increasingly worse performance on the problem. This plot collects data from a c-tuned QL instance, many other values of c have worse performance.

From the learning curve, it is plain to see that Q learning fails to learn this problem- demonstrating performance several times less efficient than the random agent, Not only do we remark on the poor performance indicated, but also note that the displayed instance is actually a lucky find- most QL instances perform much worse. We considered that the curve may indicate divergent learning, and so investigate inverted R signals and alternate critics . All such trials produce similar behavior of the QL agent. We conclude from the fact that the hand coded agent does indeed perform with high efficiency, that the QL agent is not an effective learner in this problem domain.

The next portion of the experiment batch is application of STQL to this problem. Figure 8 shows performance curves for the random agent, the hand coded agent, and the STQL agent. Our first observation is that STQL does indeed learn the problem, as evidenced by dramatically improved performance over the random agent. Also, we observe the same pattern of exponential learning curves for this problem as we saw for the taxi problem- indicating that the agent consistently does most of its learning during first representative epoch. By 'representative epoch' we mean a high entropy case, as opposed to one of low entropy, such as the goal being randomly selected just two or three cells removed from the starting location. For reference, the random agent performance displayed is the same data set as in Figure 7.

Looking to Figure 9, which represents the same data but with superior resolution, we can make further observations on the learning curve. First, we note that the STQL agent's average performance is approximately 20% worse than that of the hand coded agent. Given the poor performance of both QL and the random agent, this is reasonable performance, but begs the question of sub-optimality in our agent.

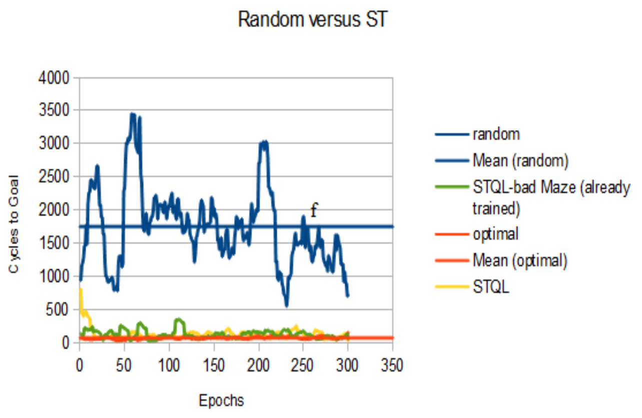

Fig. 8. Graphical representation of performance of various STQL instances as compared to the random agent, indicating that STQL is in fact learning to efficiently navigate a maze, as opposed to random walking into the goal state.

By allowing training to run over a much longer period (around 2000 epochs) we find that the performance peaks at around 15%. Looking at the critic, however, we note that as a wall follower, it does not always pick the best path (clockwise versus counter clockwise traversal, for one example of an unaccounted factor). Consequently, the exploratory aspect of the agent's decision process (the random selection) sometimes results in superior performance, which allows the variance between the exemplar and the agent to persist. This flaw in representation of the problem domain leads to a statistical non-correlation, the differences between ground truth of goal achievement and the critic create tension in the agent's learning process. However, creating a critic which solved the problem using exhaustive methods would be counter to our current goals, and perhaps be a poor trainer due to possibly implicating information which is unobservable to the agent. This situation serves to show that relatively efficient behaviors can be learned from an inexact critic. This is an emergent feature of the architecture and is fascinating in two ways. First, it shows a direct case in which a critic which does not exactly represent the problem domain can still be an effective trainer. This is useful in a situation in which the designer may not have an exact solution to the problem domain, or a precise solution is intractable. The second becomes apparent under a follow up experiment motivated by this observation.

Inspired by the quality of the agent in spite of the deviation between the critic and the problem domain, we make the following modifications to the maze to test agent's efficacy in a related, but different environment. We allow loops in the maze, so that it is no longer simply connected, and further allow non-uniform path widths. In this case, the hand coded agent fails to solve the problem, getting lost or stuck in a large number of cases. When we apply the STQL agent after training with the same critic, however, we see performance (plotted on Figure 9, also) comparable to that of the agent in the original training domain. What this shows is that the agent can successfully adapt knowledge learned from an idealized case to solve a related but more complex problem. This is highly advantageous for robotics application, as it implies that it is a possibility to train the agent in a simulation environment, and then transfer the agent to a physical system, as a method of avoiding bootstrap complications that occur in fully physical learning environments.

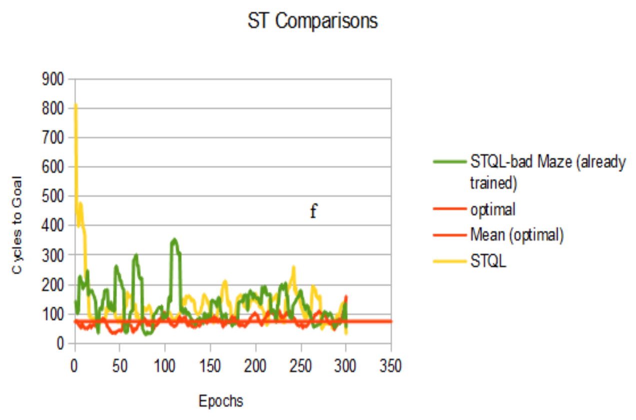

Fig. 9. Performance of STQL on the maze problem as compared to an optimal algorithm, demonstrating that STQL learns effective solutions to problems, in addition to superior performance to random and QL. Additionally, shows performance of the pre-trained agent on the non-simply connected maze which is insoluble to the 'optimal' algorithm.

This result, interesting in its own right, motivates another experiment. Based on the observation that the system demonstrates resilience to inaccuracy in the critic, we are pressed to investigate the overall robustness with respect to critic perturbations. To test this, we design a critic with an artificial inaccuracy- given a set proportion, we randomly invert the training signal. Figure 10 shows several performance curves for the STQL agent trained with error rates varying from 10% to 35%. We note that introducing error up to roughly 20% has very little effect on the agent, with destabilization beginning around 25% error, though learning is still possible at this level, and training becoming impractical at 35%. We make the observation that as the absolute error (meaning that the error of the critic measured against a theoretically optimal performer) approaches 50%, complete destabilization is expected, as the agent will then be given as much bad information as good.

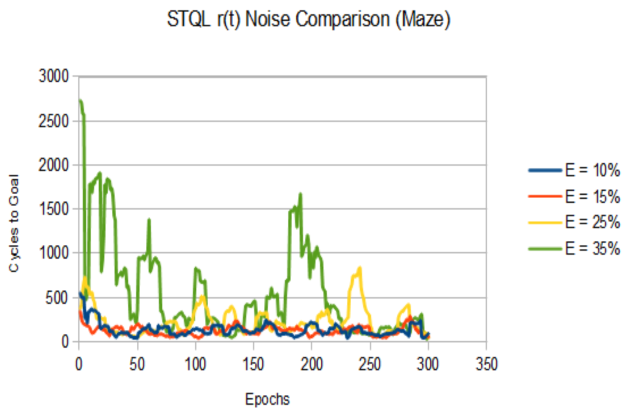

Fig. 10. Learning curve sequence showing performance with noisy reinforcement signals. Up to 15% noise in the signal produces practically negligible effect on the learning process. From a noise level of roughly 20% and up, inefficiency becomes apparent, with complete destabilization as the error rate approaches 50%.

Summarizing the results of this experiment battery, we make the following conclusions. The STQL agent is capable of learning a behavior which the QL agent cannot. It is also possible for the agent to adapt training on a problem to solve a related problem, even one in which the initial critic is not completely valid. Finally, by introducing error into the critic's training signal, we see that the STQL agent has a degree of robustness with respect to inaccurate training. These features are useful to the designer because they make the architecture amenable to environments which bear uncertainty, such as problems which do not readily admit exact solutions, or when training must be adapted to variants of the exemplar domain.

The last experiment is a proof-of-concept example, in which we apply the STQL architecture in a physical robotic agent. The target behavior is obstacle avoidance, and the domain is a flat plane on which a selection of objects are arranged. The agent takes as input two signals- a forward IR sensor, and a wheel motion sensor. From this input, the robot can detect obstacles, and whether or not it is currently moving. The robot is designed with three wheels- two steering on a servo actuated fixed axle, and one drive wheel. The action set for the robot includes setting the steering wheels to the left, right, or forward bearing, and activating the drive wheel forward or backward for a period of time. States are given by two binary variables: obstructed and stuck, indicating the presence of an obstacle or lack of movement when attempting to drive, respectively. Figure 11 shows the robot used in this experiment. The critic is designed as a state-transition table, shown in Figure 12.

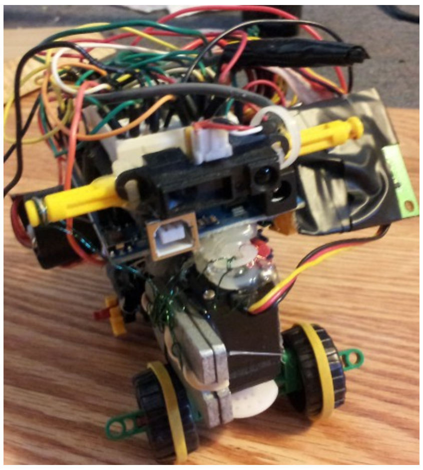

Fig. 11. The robot used in experiment battery 5. Visible are steering servo, USB programming port, balance weights, and IR sensor.

	Snew			
Sprev	UF	US	OF	OS
UF	0	0	0	0
US	1	-1	0	-1
OF	1	1	-1	-1
OS	1	1	1	-1

Fig. 12. State-transition matrix for the obstacle avoidance critic. U/O represents the state of the obstruction state variable, and F/S represents the stuck variable. Note that this table includes null training entries.

The critic is implanted internally to the robot software, applying reinforcement via a supervisory function. On this critic, we also note the first instance of a null reinforcement signal- there are entries on the table which do not confer training. This reflects the fact that we are training for a behavior that does not apply throughout the entire world set (the robot only performs avoidance when it needs to avoid something). The training then proceeds by allowing the robot to act in the environment, with the critic essentially 'looking over it's shoulder'.

Naturally, the robot at first exhibits no coherent motion, but as it wanders through the world, it begins to collide with obstacles, incurring negative feedback: for not changing it's state from obstructed to unobstructed, and for remaining in the 'stuck' state- it will observe an attempt to drive with no resulting wheel movement in certain collisions. This latter sensation was implemented to counter the cases where the robot may turn so that the IR sensor was free, but the robot was still obstructed by the obstacle. The training cycle observed took roughly five minutes for the robot to begin demonstrating coherent avoidance behavior, eventually demonstrating a single common behavior pattern of backing away from obstacles, then turning away from them (as illustrated in Figure 13). This behavior qualitatively demonstrates learning of the desired problem, mimicking a common strategy employed by human designers without any explicit representation thereof. It also demonstrates linking, as the reverse turn, then forward turn away is consistently repeated as a pattern. An interesting annotation is that the robot occasionally displays a short 'neurotic' behavior- it will turn the steering wheels in either direction before performing the avoidance behavior proper. This does not occur in every case, but often enough to be notable. We suspect this pattern is an artifact of an early random pattern, which would become less and less frequent as training progressed, based on the tuning behavior of the simulated agents.

Although notation of these qualitative features represents achievement of the goals of this experiment (which is recall, proof-of-concept for robotic application of STQL), another observation is made. In the experiment, one of the obstacles placed on the field had several long protrusions. During the course of the experiment, loose wires on the robot became tangled on these protrusions. Out of curiosity, the trial was allowed to continue to see how the agent would behave, given that it was clear that the 'obstructed' variable would not be set, as the IR sensor was not blocked sufficiently by the object. Interestingly, the agent maneuvered fruitlessly for several seconds before initiating a back and forth behavior that eventually disentangle it from the object. This observation mirrors the performance in experiment four, in which the agent was able to learn in the altered maze circumstance. Again, these are qualitative observations, which poses a difficulty for interpretation that is compounded by the issues associated with physical testing, but it provides a good extension of previous observation into a physical computing context.

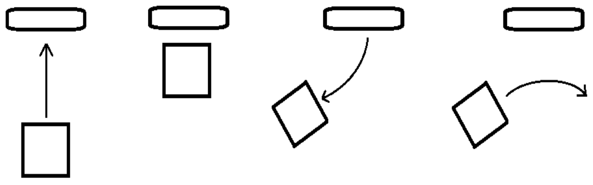

Fig. 13. Illustration of the final procedure for the robot, showing general motion pattern, not including the independent wheel movement actions, for clarity.

To close this section, we summarize our results. First, we noted speed comparability to QL on basic linearly inseparable problems, the XOR and 2XOR problems. We also observed increased processing efficiency as the size of the domain was increased by comparing the XOR learning curves to the 2XOR curves. We evaluated performance on a more dynamic problem by comparing STQL to QL on the taxi problem, noting first the similarity of learning speed between the two, and the exponential trend in the STQL learning curve. In the fourth experiment battery, we demonstrated efficacy of STQL on the maze domain, a problem which was intractable to QL, as well as establishing learning by comparison to the random agent. We then, inspired by the discovery of inexactness in the critic, found that the STQL agent could adapt learning to a new, related problem that the critic could not solve. This motivated us to investigate how the corruption of the learning signal effects learning, finding that the agent displayed robustness to a moderate amount of error- roughly 20%. Lastly, we implemented STQL in an embedded system physical agent, as proof of concept. Here, we qualitatively noted that the learning speed was acceptable for physical system learners, and observed first hand the development of a linked behavior. We also saw that the agent mirrored the capabilities seen in the maze experiment, learning to solve a related, but different problem.

V. Conclusions

We would like to conclude by collecting observations, mentioning some drawbacks we are addressing, and discussing future work. Inspired by operant conditioning research, we describe an architecture which establishes links between actions via recurrence, and learns patterns on those links by temporally associated reinforcement feedback. We illustrated how these procedures create historical dependence in the learner, as well as how chains can be developed from them. Additionally, we showed via a sequence of experiments that the architecture learns efficiently, and is performance comparable to standard Q Learning- so as to establish efficacy and efficiency.

With the maze problem, we were able to demonstrate learning of behaviors that are inaccessible to a Q Learning agent. Furthermore, our experiments also illustrated the flexibility of the STQL agent by showing learning in multiple problem domains, and by transferal of effectiveness to related problems in two cases. We also note that the training method is variable, as some critics were implemented as state-transition matrices, and some as hard-coded goal achievement supervisors.

Currently, the most critical drawbacks we encounter include the need to partition the input and output space, lack of state space reduction, and critic design. To address the former, we are working to implement a neural form of the architecture. We have met with some success with this approach, however we currently lack sufficient support to present it here. The second issue may possibly be resolved with extant abstraction algorithms, however we have not yet attempted implementation thereof- for the purposes of this research, it was unnecessary.

The issue of critic design is the more complex of the problems we mention here. Although we have shown that an inexact critic can still be an effective teacher, the issue of accuracy is still a persistent concern- a better critic still correlates with a better agent. Our first tactic to improve the situation is attempting to use another learning algorithm to develop a model the environment, training it based on net improvement in agent quality. This would allow the critic to grow along side the agent, developing its reinforcement policy in parallel.

We are also considering modifying the decision engine structure to operate on a winner-take-all policy. By doing so, we believe that the influence of randomness as an effective policy (and in low-information cases, it can be highly effective) can be reduced. However, we do suspect that this will reduce the flexibility and generality of learning somewhat, due to reduction in exploration. Experimentation along these lines will most likely be the effective way to guide these investigations.

For future work, we have several directions in which we suspect fruitful work may be done. Our first interest is with using the STQL agent to learn sub-module behaviors efficiently for implementation within higher-order processes. This seems to be the most obvious use for the architecture. We will also be looking into casting a multilayered agent, in which sub-behaviors are generated via STQL, and another STQL implementation has as it's output actions behaviors generated by lower agents.

The second application we intend to pursue is the use of the modified architecture in an adaptive control system. We hypothesize that creating a critic which rewards an agent by examining error metrics, we can train an STQL agent to emulate a control system. Some basic preliminary examples seem to suggest this may be feasible, in particular work on using the architecture to teach a wall following behavior bears similarities to the control problem.

Our last remark is on the most abstract goal we envision for the architecture currently. In examining the literature on sequential behaviors, we note the similarity between sequences of actions an sequential processing. This leads us to theorize that it may be possible to apply the sequence training to an abstract information processing problem, such as solving an algebraic equation or sorting a list. Though to do this, the problem expression would have to be very well founded, and the critic would require significant development to express concepts such as 'simpler equations' or 'more ordered lists' in a well-conditioned form for training. We plan to begin investigating some simple such operations shortly.

References

[1] B. F. Skinner, "The Behavior of Organisms:An Experimental Analysis", New York: Appleton-Century-Crofts , 1938

[2] M. Hauskrecht, N. Meuleau, C. Boutilier, L. Kaelbling, & T. Dean, “Hierarchical solution of Markov decision processes using macro-actions”. Tech. rep., Brown University, Department of Computer Science, Providence, RI. 1998.

[3] S. Mahadevan & J. “Automatic programming of behavior based robots using reinforcement learning” ,IBM T.J. Watson Research Center,Yorktown heights, NY; AAAI Proceedings, 1991

[4] D. Touretzky & L. Saksida “Operant conditioning in skinnerbots”, 1997

[5] B. Blumberg, P. Todd, P. Maes “No Bad Dogs: Ethological Lessons for Learning in Hamsterdam”; MIT Media Lab E15-305, 20 Ames St. Cambridge Ma.; Max Planck Inst. for Psychological Research, Center for Adaptive Behavior and Cognition, Leopoldstrasse 24, 80802 Munich Germany, 1996

[6] J. Bongard, “Behavior Chaining: Incremental Behavior Integration for Evolutionary Robotics”, University of Vermont, Burlington, VT, 2008