Advancing Motivated Learning with Goal Creation

James Graham[1] and Janusz A. Starzyk[1,2]
[1]School of Electrical Engineering and Computer Science
Ohio University, Athens, OH, USA
[2]University of Information Technology and Management
Rzeszow, Poland
{jg193404, starzykj}@ohio.edu

Zhen Ni and Haibo He
Electrical, Computer, and Biomedical Engineering
University of Rhode Island, Kingston, RI, USA
{ni,he@ele.uri.edu}

***Abstract*—This paper reports improvements to our Motivated Learning (ML) model. These include modifications to the calculation of need/pain biases, pain-goal weights, and how actions are selected. Resource based abstract pains are complemented with pains related to desired and undesired actions by other agents. Probability based selection of goals is discussed. The minimum amount of desired resources is now set automatically by the agent. Additionally, we have presented several comparisons of Motivated Learning performance against some well-known reinforcement learning algorithms.**

Keywords—motivated learning; reinforcement learning; goal creation; pain signals; desired resources.

I. INTRODUCTION AND BACKGROUND

This paper presents advancements of our Motivated Learning algorithm. Specifically, we discuss calculation of bias signals, consider desired and undesired situations, modify interconnection weight calculations, use probabilistic goal selection, and determine desired resource levels. Additionally, we compare our algorithm with various implementations of Reinforcement Learning (RL) algorithms.

Reinforcement Learning and Motivated Learning (ML) share several similarities in how they function. They both seek rewards, however, ML develops internal motivations [1], so only its designer specified rewards can be observed and measured. A ML machine also obtains intrinsic rewards, and what is rewarded depends only on its internal state and the mechanisms that created its internal needs. Both approaches can use artificial curiosity. An Intelligent Adaptive Curiosity algorithm (IAC) was proposed by Oudeyer et al. [2], and our own agent first utilized curiosity in [3].

The IAC algorithm allows a robot to function in continuous, noisy, inhomogeneous environments, and allows autonomous self-organization of behavior toward increasingly complex behavior patterns. The problem with IAC is that while it eliminates some of the issues with purely curiosity based learning, it is still learning without purpose. The agent will try to learn everything useful, rather than optimize for specific skills and higher levels of performance. This is where Motivated Learning comes in, to provide purpose and direction of the agent's development.

Pfeifter and Bongrand [4], believe that an agent's motivations should emerge as part of the development process. Merrick pointed out that RL agents do not have internal drives to maintain their resources within an acceptable range, and that the designer cannot realistically determine all useful goals ahead of time [5]. These issues were addressed in [6] where the motivated learning concept with internal goal creation was introduced. In a similar development Merrick described motivated reinforcement learning (MRL) and used motivated exploration in video games [5].

In our current ML implementations, we consider a symbolic implementation of a motivated agent that uses concept neurons to represent perceptions, pains, goals and motor actions. A neural network that links sensory inputs to motor outputs emerges from the learning process. In particular, nodes that represent perceptions, abstract pains, goals, and learned motor actions are added dynamically as the agent learns how to interact with the environment. Full implementation of Motivated Learning schema requires symbol grounding and learning to execute complex actions through sequences of motor control. These however are separate problems that are being worked out in robotics and computational intelligence research programs [7]-[10]. As new concept nodes are added they are linked by association links with variable weights. Actual actions are chosen via a combination of the aforementioned weights and heuristics based on our OML algorithm [11]-[12].

In the following section we present how an ML agent learns to adjust weights that associate pain and goal neurons in the neural network. Then we describe changes to bias calculation, weight adjustment, and goal selection. Following this, we show how the aforementioned changes affect the ML agent's operation. Section III discusses setting the desired level of resources and distances to other agents. Next, we show how our Motivated Learning algorithm fairs against several Reinforcement Learning algorithms, followed by the paper's conclusion.

II. ENHANCEMENTS TO THE ML ALGORITHM

Next we discuss modifications to our Motivated Learning algorithm. We illustrate these enhancements in section IV.

A. How to calculate bias signals

In our motivated learning algorithm bias signals are used to determine the level of pain or need associated with a particular concept (resource availability or an action performed by another agent). Biases indicate the likelihood of running out of resources needed or of facing a hostile action by another agent. There are several ways in which bias can be calculated. One method is a simple logarithmic approach,

This research was supported by The National Science Centre, grant No. 2011/03/B/ST7/02518, the National Science Foundation under grant ECCS 1053717, and the Army Research Office under grant W911NF-12-1-0378.

$$B_i = \gamma \cdot \frac{\varepsilon + R_d(S_i)}{\varepsilon + R_c(S_i)} \quad (1)$$

where R_c is the current resource value, R_d is the desired (or initial) resource value, s_i represents the resource under consideration, and ε is a small positive number. Eqn. 1 is useful because it produces a large bias as a desired resource approaches zero. On the down side it does not go to zero when $R_c = R_d$.

Originally, the initial state was considered to be the desired state of the agent; however, the initial state of a resource is not necessarily the optimum state for the agent. The agent should be able to set said desired state all by itself. We discuss how this might be done is section III.

Another way to calculate bias is using:

$$B_i = \left| log2 \left(\frac{\varepsilon + R_c(S_i)}{\epsilon + R_d(S_i)} \right) \right| \quad (2)$$

Eqn. 2 works well when the probability of a resource is used (as was the case in our initial experiments).

A third variant uses slightly more complex computations to deliver a smooth transition around the desired resource level:

$$B_i = -ln \frac{R_c(S_i)}{R_d(S_i)} + \frac{R_c(S_i)}{R_d(S_i)} - 1 \quad (3)$$

B. Bias based on availability

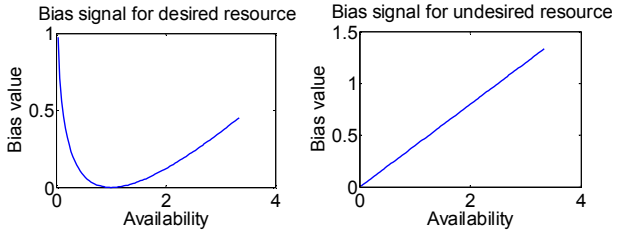
Bias signals (1)-(3) are based on desired resource availability. But in the general case a resource can either be desired or undesired. In addition, actions by other agents can be desired or not. A single equation can be used for both resource and action biases for both desired and undesired cases. Assume that A represents availability ($A=R_c/R_d$ for resource biases, where R_d represents desired level of resource observable in the environment, $A=1/(d_c/d_d+1)$, where d_d is a desired (a comfortable) distance to another agent. Then the bias signal is obtained from:

$$B_i = -(1 + \delta_i) * (lnA(s_i) + 1) + 2 * A(s_i) \quad (4)$$

$$\delta_i = \begin{cases} 1 \; for \; desired \; resource \; or \; action \\ -1 \; for \; undesired \; resource \; or \; action \end{cases} \quad (5)$$

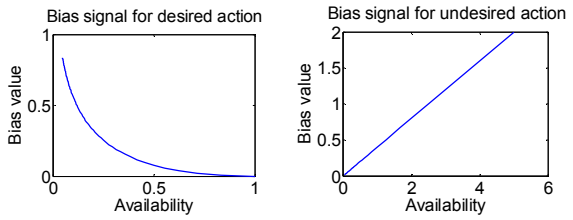
$$A(s_i) = \begin{cases} \frac{R_c}{R_d} & for \; resource \; bias \\ \frac{1}{\frac{d_c}{d_d} + \frac{1+\delta_i}{2}} & for \; action \; bias \end{cases} \quad (6)$$

Depending on desirability or resource/action (4) produces different results as shown in Figs. 1-2.



a) b)

Fig. 1 Bias signal for a) desired resource b) undesired resource.



a) b)

Fig. 2 Bias signal for a) desired action b) undesired action.

Observing Figs. 1b and 2b, it can be seen that biases for both undesired resources and undesired actions appropriately follow the same increasing trend as the associated action/resource becomes increasingly "available". Since bias has a direct effect on the pain level, it follows that an increase in the "availability" of an undesired action/resource and corresponding increase in bias level increases the associated pain. Bias signals for both desired resources and desired actions in Figs 1a and 2a, go to zero as their availability reaches a desired level equal to 1. That bias increases for desired resources as they pass the desirability threshold. This simulates the "too much" of something effect.

C. Learning to select goals

Our Motivated Learning agent selects which actions to take based on pain levels and pain-goal weights (w_{PG}). These weights are adjusted dynamically through learning as discussed in [13] using:

$$w_{PG} = w_{PG} + \delta_p \cdot \Delta_a \quad (7)$$

In this equation the magnitude of w_{PG} is adjusted based on the value of Δ_a, which is calculated based on the previous w_{PG} weight and the current resource levels. δ_p is an integer value that can be zero, positive, or negative depending on how the associated pain/need was effected by the last action, and

$$\Delta_a = \mu_g \cdot min\left(|\alpha_g - w_{PG}|, w_{PG}\right) \quad (8)$$

$$\mu_g = \mu_0 \cdot \left[1 - \frac{2}{\pi} \cdot atan\left(10 \cdot \left(\frac{R_c(s_i)}{R_d(s_i)}\right)^{\delta_i}\right)\right] \quad (9)$$

$\alpha_g \leq 1$ and represents the "ceiling" for the w_{PG} weights, μ_0 is a constant, (set in our work to 0.3), δ_i represents whether the resource specified by s_i is desired '1', undesired '-1', or unknown '0'.

D. Probabilistic Goal Selection

In a probabilistic goal selection approach, goals are selected using probability values based on the current w_{PG} weights:

$$p_{PGi} = \frac{w_{PGi}}{\sum_{i=1}^{n_{PG}} w_{PGi}} \quad (10)$$

In this case there is no need to limit weight values as in (8) since we can assume (after normalization)

$$w_{PGi} = p_{PGi} \leq 1, \; and \; \sum_i p_{PGi} = 1 \quad (11)$$

Using (7)-(9) for probability based goal selection faced the problem of weight saturation to α_g, which resulted in all probabilities approximating the same value for successful goals independently of $\mathrm{atan}(ds/d\hat{s})$.

A modification of (7) differentiates between useful actions depending on their efficiency. Initial values of w_{PG} weight are set to $1/n_{PG} \pm \epsilon$, where n_{PG} is the number of w_{PG} weights (instead of a $0.5 \pm \epsilon$). After each time a goal is triggered by a specific pain signal its corresponding w_{PG} weight is adjusted as follows:

$$w_{PG} = min\left(\frac{3}{\pi}\text{atan}\left(\frac{ds}{d\hat{s}}\right), w_{PG} * \mu_g * \frac{4}{\pi}\text{atan}\left(\frac{ds}{d\hat{s}}\right)\right). \quad (12)$$

$\frac{ds}{d\hat{s}}$ is the rate of change of the lower level resource s as a result of using higher level resource $\hat{s}$ observed by the agent.

$$\mu_g = 1 + \left(1 - \frac{2}{\pi}\text{atan}\,(10/B)\right) * \left(\alpha_f{}^{-\delta_i} - 1\right), \quad (13)$$

where $\alpha_f < 1$ (recommended value is around 0.7) . As a result w_{PG} weights saturate at $(3/\pi)\text{atan}\,(ds/d\hat{s})$. Thus more efficient ways are more likely to be chosen in the probabilistic goal selection.

III. SETTING DESIRED RESOURCE LEVELS

Desired values should be set according to the agent's needs. We touched on this briefly in one of our earlier papers [14]. To set desired resource levels, we start with the need for the primitive resource (e.g. the energy level). This level R_{dp} is known and is set by the environment. Another environment set variable is the rate of use of the desired resource Δ_p(for instance how quickly the energy is lost by the agent). All desired levels for other resources (related to abstract pains) are set by the agent. These levels are established by the agent only for these resources that the agent cares about.

In order for the agent to be successful in all of its tasks the frequency of performing various tasks cannot be too great as its time is limited. In the simplest case, consider that each task takes one unit of time (equivalent to a single iteration). If a primitive resource will be exhausted in n_s iterations the agent must perform an operation to restore this resource with the frequency $f_s = \frac{1}{n_s}$. Considering all actions that agent performs its operations can be successful only if

$$\sum_i f_{si} < 1 \quad (14)$$

where the summation is for all resources that the agent needs to restore and all actions it must perform to avoid hostile activities by other agents.

For all primitive resources the restoration frequency is predetermined from

$$n_s = \frac{R_{ds}}{\Delta_s} = \frac{1}{f_s} \quad (15)$$

Restoration frequencies of other resources can be set arbitrarily provided that (7) is satisfied. Notice that in order to obtain the restoration frequency of a higher level resource we must consider the restoration frequency of the lower level resource it helps to restore, thus we have

$$f_{\hat{s}} = f_s * \frac{\Delta_{\hat{s}}}{R_{d\hat{s}}} < f_s \quad (16)$$

The largest restoration frequency for the higher level resource equals to the restoration frequency of the resource it helps to restore, and in such case $R_{d\hat{s}} = \Delta_{\hat{s}}$ and we need to restore the higher level resource each time after we use it. This seems to be an extreme situation, although the required resource level will be very low. However, when the agent has multiple goals we may easily violate (14). This setting may be used if we want to teach the agent proper response in a shortest possible time.

If (14) is satisfied for all primitive needs we can always set desired values of higher level resources $R_{d\hat{s}}$ in such a way that (14) holds. This may require high values of $R_{d\hat{s}}$. In general to set $R_{d\hat{s}}$ in an optimized fashion, we need to solve the optimization problem:

$$F(\alpha_1, \alpha_2, \ldots, \alpha_m) = min \sum_{i=1}^{m} \frac{1}{\alpha_i} \quad (17)$$

Subject to constraints

$$0 \le \alpha_i \le 1, \qquad i = 1, \ldots, m \quad (18)$$

and

$$\sum_{i=1}^{p} f_{si}\left(1 + \alpha_{i1}\left(1 + \alpha_{i2}\left(1 + \ldots + \alpha_{n_i}\right)\right)\right) < 1 \quad (19)$$

where p is the number of primitive, α_{i1} represents the resource $\hat{s}_{i1}$ that is needed to restore s_i and

$$\alpha_{i1} = \frac{\Delta_{\hat{s}_{i1}}}{R_{d\hat{s}_{i1}}} \quad (20)$$

is a relative frequency of the higher level resource to be determined, which tells us how many times we can use it before it is gone and needs to be replenished.

In a similar way α_{i2} represents the resource $\hat{s}_{i2}$ that is needed to restore $\hat{s}_{i1}$ and so on, and where $\hat{s}_{ni}, \ldots, \hat{s}_{i2}, \hat{s}_{i1}, s_i$ is a dependence path from the most abstract resource $\hat{s}_{ni}$ to the primitive resource s_i.

Once α_i is known, we can determine the desired level of each higher level resource using

$$R_{d\hat{s}_i} = \frac{\Delta_{\hat{s}_i}}{\alpha_i} \quad (21)$$

The environment set rules establish derivative $\frac{ds_i}{d\hat{s}_i}$ which can be used to compute $\Delta_{\hat{s}_i}$ in order to restore the level resource to its desired value R_{ds}

$$\Delta_{\hat{s}_i} = \frac{R_{ds_i}}{\frac{ds_i}{d\hat{s}_i}} \quad (22)$$

thus

$$\mathrm{R}_{\mathrm{d}\hat{\mathrm{s}}_\mathrm{i}} = \frac{\mathrm{R}_{\mathrm{ds}_\mathrm{i}}}{\alpha_\mathrm{i} * \frac{\mathrm{ds}_\mathrm{i}}{\mathrm{d}\hat{\mathrm{s}}_\mathrm{i}}} \quad (23)$$

To solve the optimization problem (17) we use the Matlab function *fmincon* with 'sqp' or sequential quadratic programming algorithm, which rigidly respects bounds, and is robust in handling *inf* and *nan* values.

To illustrate setting the desired resource level as discussed in this section let us consider the following example:

Example 1:

Let us assume that we have 3 primitive resources with the need for the primitive resources equal to:

$$R_{d1} = 20, \quad R_{d2} = 12, \quad R_{d1} = 21,$$

and the rate of use of the primitive resources equal to

$$\Delta_1 = 4, \qquad \Delta_2 = 2\,, \qquad \Delta_3 = 3$$

Thus, primitive resource restoration frequencies are

$$f_{s1} = \frac{\Delta_1}{R_{d1}} = \frac{1}{5},\ f_{s2} = \frac{\Delta_2}{R_{d2}} = \frac{1}{6},\ f_{s3} = \frac{\Delta_3}{R_{d3}} = \frac{1}{7}.$$

In addition, we have 7 higher level resources that can be used to restore these primitive ones and we would like to set desired resource values in such way that the agent learns in an optimized way, and yet is capable of handling all the tasks.

Also assume the following dependence paths from the most abstract resource to the primitive resource s_i:

$$s_6, s_5, s_4, s_1$$

$$s_8, s_7, s_2$$

$$s_{10}, s_9, s_3$$

And the lower level resource restoration rates with respect to the utilization rate of the higher level resource are as follows:

$$\frac{ds_1}{ds_4} = 3, \qquad \frac{ds_4}{ds_5} = 2.5, \qquad \frac{ds_5}{ds_6} = 4,$$

$$\frac{ds_2}{ds_7} = 2, \quad \frac{ds_7}{ds_8} = 5, \quad \frac{ds_3}{ds_9} = 5, \quad \frac{ds_9}{ds_{10}} = 1.5$$

Thus, our optimization problem can be formulated as follows:

$$F(\alpha_4, \alpha_5, \dots, \alpha_{10}) = min \sum_{i=4}^{10} \frac{1}{\alpha_i}$$

$$\alpha_i \le 1, \qquad i = 4, \dots, 10$$

and

$$f_{s1}\left(1 + \alpha_4\left(1 + \alpha_5(1 + \alpha_6)\right)\right) + f_{s2}\left(1 + \alpha_7(1 + \alpha_8)\right) + f_{s2}\left(1 + \alpha_9(1 + \alpha_{10})\right) < 1$$

To solve this, we wrote an objective function and a constraints function in Matlab and invoked a constrained optimization routine to obtain the optimized parameter values equal to:

$$x = [0.398 \quad 0.686 \quad 1.00 \quad 0.477 \quad 0.969 \quad 0.511 \quad 1.00]$$

and the objective function value equal to 11.0673.

In this solution α_{3+i} corresponds to *x(i)*, thus using (23) we have

$$R_{d4} = \frac{R_{d1}}{\alpha_4 * \frac{ds_1}{ds_4}} = \frac{5}{0.3965 * 3} = 16.8138$$

In a similar way we will obtain

$$R_{d5} = 9.8082, R_{d6} = 2.4521, R_{d7} = 12.5865,$$
$$R_{d8} = 2.5984, R_{d9} = 8.2208, \quad R_{d10} = 5.4805.$$

Thus, we can determine the desired resource values by using the provided hierarchy and other provided data in conjunction with the optimized values to calculate the desired resource levels.

Network dependencies

An acyclic case

The previously discussed case possessed strict hierarchical dependencies in which abstract resources were stacked against lower level resources. In a more general case we may consider network dependencies between resources as shown on Fig. 3.

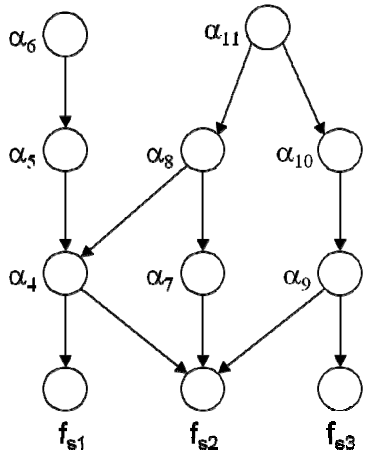


Fig. 3 Abstract resource dependencies.

Fig. 3 illustrates an acyclic case where we have 3 primitive resources that are used with frequencies f_{s1}, f_{s2}, and f_{s3} respectively and 8 abstract resources with dependencies illustrated by downward pointing arrows.

To establish the optimum level of desired abstract resources we can solve the optimization problem as before with only slightly modified constraints

$$F(\alpha_1, \alpha_2, \dots, \alpha_m) = min \sum_{i=1}^{m} \frac{1}{\alpha_i} \tag{24}$$

Subject to constraints

$$0 \le \alpha_i \le 1, \qquad i = j + 1, \dots, m \tag{25}$$

and sum of all frequencies is less than 1

$$\sum_{i=1}^{d} f_{si} < 1 \tag{26}$$

where *j* is the number of primitive resources and each abstract node frequency $f_{\hat{s}}$ is obtained by multiplying $\alpha_{\hat{s}}$ by the sum of frequencies of lower level nodes that directly depend on it

$$f_{\hat{s}} = \alpha_{\hat{s}} * \sum_{i=1}^{\hat{s}} f_{si} \tag{27}$$

Since in this case

$$\alpha_{\hat{s}_i} = \frac{\Delta_{\hat{s}_i}}{R_{d\hat{s}_i}} = \frac{\sum_{i=1}^{\hat{s}} \frac{R_{ds_i}}{ds_i}}{R_{d\hat{s}_i}}, \quad (28)$$

where summation is over all lower level resource that are dependent on resource $\hat{s}_i$, so the desired level of abstract resource can be computed from

$$R_{d\hat{s}_i} = \frac{\sum_{i=1}^{\hat{s}} \frac{R_{ds_i}}{ds_i}}{\alpha_{\hat{s}_i}} \quad (29)$$

A general case

In general, a resource dependency graph such as that shown in Fig. 3 may have cyclical dependencies and we need to modify the way the desired resource level is established. In addition, when a resource can be restored in several different ways, we may use probabilities for selecting actions that lead to restoration of the resource. The two considerations can be combined in a general case of desired resource optimization.

To establish the optimum level of desired abstract resources we can solve the optimization problem as before

$$F(\alpha_1, \alpha_2, \dots, \alpha_m) = min \sum_{i=1}^{m} \frac{1}{\alpha_i} \quad (30)$$

Subject to constraints

$$0 \le \alpha_i \le 1, \qquad i = 1, \dots, m \quad (31)$$

and sum of all frequencies is less than 1

$$\sum_{i=1}^{d} f_{si} < 1 \quad (32)$$

where each abstract node frequency $f_{\hat{s}}$ is obtained by solving the following linear equation:

$$diag\left(\frac{1}{\alpha_{\hat{s}}}\right) * F_{\hat{s}} = P_{\hat{s}} * F_{\hat{s}} + P_s * F_s \quad (33)$$

where $F_{\hat{s}}$ is a vector of abstract resource frequencies that need to be evaluated, $diag(1/\alpha_{\hat{s}})$ is a diagonal matrix of relative frequencies for higher level resources to be determined in the optimization process, $P_{\hat{s}}$ is a $m \times b$ matrix of action related resource selection probabilities to restore abstract resources on a lower level, where m is the total number of abstract resources, and b is a bottom part of abstract resources (excluding top level inexhaustible resources). Similarly, F_s is a known vector of primitive resource frequencies and P_s is $m \times p$ matrix of action related resource selection probabilities to restore the primitive resources and p is the number of primitive resources.

Each element p_{ij} of column j of P_s and $P_{\hat{s}}$ contain sum of probabilities of actions using resource i to restore resource j. Since sum of probabilities of all actions to restore resource j is equal to 1, then each column of P_s and $P_{\hat{s}}$ adds to 1.

We can further divide (31) to separate top level (inexhaustible resources) from other resources.

$$\begin{bmatrix} diag\left(\frac{1}{\alpha_{\hat{s}_t}}\right) & 0 \\ 0 & diag\left(\frac{1}{\alpha_{\hat{s}_b}}\right) \end{bmatrix} * \begin{bmatrix} F_{\hat{s}_t} \\ F_{\hat{s}_b} \end{bmatrix} = \begin{bmatrix} P_{\hat{s}_t} \\ P_{\hat{s}_b} \end{bmatrix} * F_{\hat{s}_b} + \begin{bmatrix} P_{s_t} \\ P_{s_b} \end{bmatrix} * F_s \quad (34)$$

Where $diag(1/\alpha_{\hat{s}_t})$ and $diag(1/\alpha_{\hat{s}_b})$ are diagonal matrices of relative frequencies for top and bottom higher level resources, respectively. And $F_{\hat{s}_t}$ and $F_{\hat{s}_b}$ are vectors of frequencies of use for top and bottom resources, respectively. In a similar way we defined $P_{\hat{s}_t}$ and $P_{\hat{s}_b}$ as well as P_{s_t} and P_{s_b} as submatrices of $P_{\hat{s}}$ and P_s that correspond to the top and bottom part of abstract and primitive resources. In (32) only F_s is known and $F_{\hat{s}_t}$ and $F_{\hat{s}_b}$ must be evaluated.

This linear equation can be solved in two steps, first to obtain $F_{\hat{s}_b}$ using

$$diag\left(\frac{1}{\alpha_{\hat{s}_b}}\right) * F_{\hat{s}_b} = P_{\hat{s}_b} * F_{\hat{s}_b} + P_{s_b} * F_s \quad (35)$$

so

$$F_{\hat{s}_b} = \left(diag\left(\frac{1}{\alpha_{\hat{s}_b}}\right) - P_{\hat{s}_b}\right)^{-1} * P_{s_b} * F_s \quad (36)$$

after which $F_{\hat{s}_t}$ can be obtained directly from

$$diag\left(\frac{1}{\alpha_{\hat{s}_t}}\right) * F_{\hat{s}_t} = P_{\hat{s}_t} * F_{\hat{s}_b} + P_{s_t} * F_s \quad (37)$$

Since we can estimate that

$$\Delta_{\hat{s}_i} = \sum_{i=1}^{\hat{s}} p(\hat{s}_i)_{s_i} * \frac{R_{ds_i}}{\frac{ds_i}{d\hat{s}_i}} \quad (38)$$

where $p(\hat{s}_i)_{s_i}$ stands for the probability of using resource $\hat{s}_i$ to restore resource s_i.

Then after solving the optimization problem, the desired resource level of various abstract resources can be obtained as:

$$R_{d\hat{s}_i} = \frac{\Delta_{\hat{s}_i}}{\alpha_i} \quad (39)$$

Since the solution of (38) may depend of $R_{d\hat{s}_i}$ that needs to be determined from (39), we have to solve this linear problem in a similar way as (34) by subdividing it into smaller parts. Let us first define a matrix

$$\chi(\hat{s}_i, s_i) = \frac{p(\hat{s}_i)_{s_i}}{\alpha_i * \frac{ds_i}{d\hat{s}_i}} \quad (40)$$

Desired resource values can be obtained from

$$R_{d\hat{s}_i} = \chi(\hat{s}_i, s_i) * R_{ds_i} \quad (41)$$

Next we partition (41) to separate top level resource $R_{d\hat{s}_t}$ (inexhaustible) bottom level abstract resources $R_{d\hat{s}_b}$ and primitive resources $R_{d\hat{s}_p}$ (with given desired resource levels):

$$\begin{bmatrix} R_{d\hat{s}_t} \\ R_{d\hat{s}_b} \end{bmatrix} = \begin{bmatrix} \chi(\hat{s}_i, s_i)_{tb} & \chi(\hat{s}_i, s_i)_{tp} \\ \chi(\hat{s}_i, s_i)_{bb} & \chi(\hat{s}_i, s_i)_{bp} \end{bmatrix} * \begin{bmatrix} R_{d\hat{s}_b} \\ R_{d\hat{s}_p} \end{bmatrix} \quad (42)$$

We can solve

$$R_{d\hat{s}_b} = [\chi(\hat{s}_i, s_i)_{bb} \quad \chi(\hat{s}_i, s_i)_{bp}] * \begin{bmatrix} R_{d\hat{s}_b} \\ R_{d\hat{s}_p} \end{bmatrix} \tag{43}$$

To obtain $R_{d\hat{s}_b}$ from $R_{d\hat{s}_p}$

$$R_{d\hat{s}_b} = (1 - \chi(\hat{s}_i, s_i)_{bb})^{-1} * \chi(\hat{s}_i, s_i)_{bp} * R_{d\hat{s}_p} \tag{44}$$

And then compute $R_{d\hat{s}_t}$

$$R_{d\hat{s}_t} = [\chi(\hat{s}_i, s_i)_{tb} \quad \chi(\hat{s}_i, s_i)_{tp}] * \begin{bmatrix} R_{d\hat{s}_b} \\ R_{d\hat{s}_p} \end{bmatrix} \tag{45}$$

Example 2:

To illustrate this general case let us consider the resource dependencies graph shown in Fig. 4.

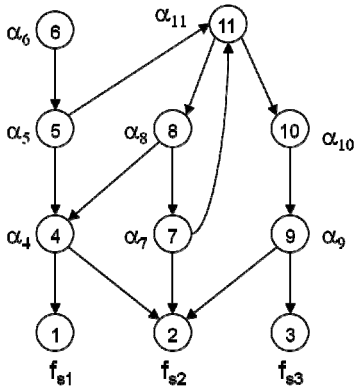


Fig. 4 Graph with cyclical resource dependencies.

Assume that related resource-to resource probability matrix P is as follows:

$$P = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0.2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.4 & 0 & 0 & 0 & 0 & 0 & 0 & 0.5 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0.5 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.5 \\ 0 & 0 & 0 & 0.6 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0.3 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \end{bmatrix}$$

where P_{ij} shows probability of using resource i to restore resource j, and the matrix of derivatives of the dependent lower level resource over the higher level resource DS is

$$DS = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 3 & 2.5 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 4 & 0 & 0 & 0 & 0 & 0 & 0 & 3 \\ 0 & 0 & 0 & 0 & 2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 8 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2.2 \\ 0 & 0 & 0 & 5 & 0 & 0 & 1.5 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 3 & 7 & 0 & 0 & 0 & 0 & 0 & 2.4 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & 0 & 2.7 & 0 \end{bmatrix}$$

$DS(i,j)$ indicates derivative of j wrt. i. Notice that this order was flipped as compared to matrix P for easier calculation of $\chi(\hat{s}_i, s_i)$. In the resource dependencies graph resources 1, 2 and 3 are primitive resources, and resource 6 is a top level resource. The desired level of primitive resources is known

$$R_{d\hat{s}_p} = \begin{bmatrix} 20 \\ 12 \\ 21 \end{bmatrix}$$

and with $\alpha_{\hat{s}_t} = [0.3686 \quad 0.7620 \quad 0.5446 \quad 0.5391 \quad 0.3934 \quad 1.0000 \quad 0.6581 \quad 0.5318]$

We can solve (37) and (38) to get the desired level of all resources

$R_{d\hat{s}_b} =$

$[\ 20.691 \quad 9.380 \quad 14.238 \quad 30.443 \quad 4.200 \quad 2.659 \quad 30.472\ ]'$

and

$$R_{d\hat{s}_t} = [\ 8.6127].$$

IV. Simulation Results

In this section we provide results showcasing how the changes presented in Sections II-III affect the ML algorithm.

A. Desirability calculations

We spend a significant amount of time outlining how and agent can determine the desired level of the resources it its environment. In Fig. 5 we show the actual adjustments of the desired resource levels as they occur. At the beginning, both graphs show the initial state of resources and hence, the initial desirability level. However, once the agent discovers that a particular resource is useful and collects the necessary information on its properties and the resource "above" it, the agent can adjust the desirability level as discussed in section III.

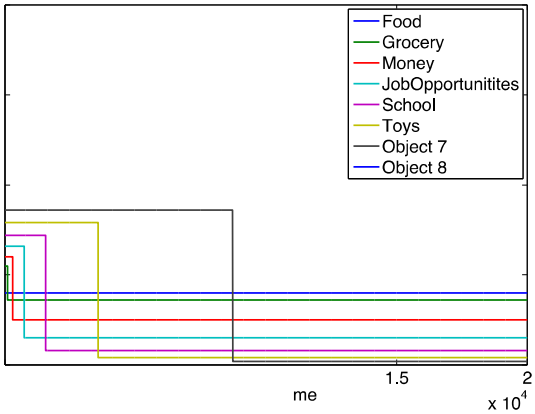


Fig. 5. Resource desirability adjustments without random action selection.

B. Probabilistic goal selection

Figure 6, shows the adjustment of w_{BP} weights of the agent over time. It was decided to show these weights since they only grow in value when their associated resource is known to effect the agent, and are hence, a good indicator of when the agent "discovers" the usefulness of a resource.

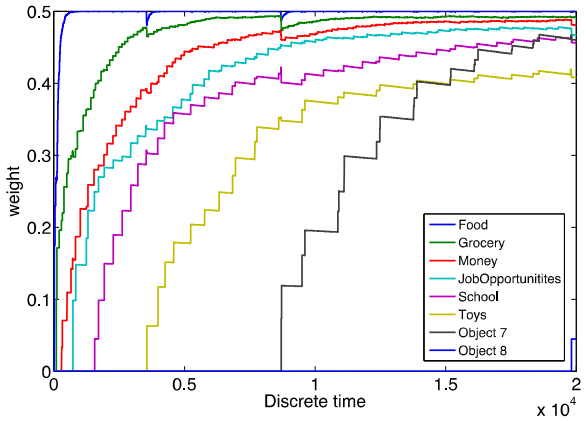


Fig. 6. w_{BP} weights without probabilistic goal selection.

Fig. 7 shows the effect of probability based goal selection on w_{BP} weight adjustment. While significantly noisier due to the effect of probabilistically choosing more "incorrect" actions, the w_{BP} weights of Fig. 7 indicate that agent is able to discover the usefulness of resources significantly earlier in the simulation when using probabilistic goal section.

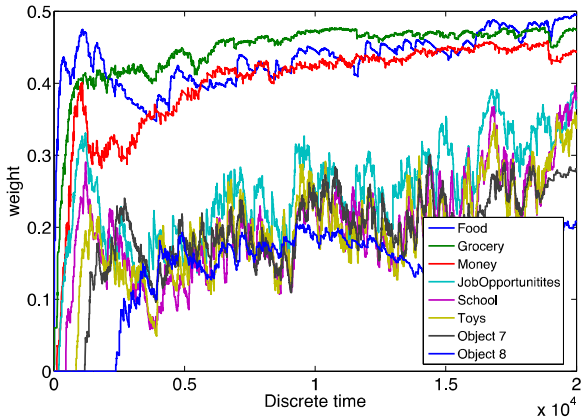


Fig. 7. w_{BP} weights with probabilistic goal selection.

Aside from the advantage of discovering useful resources earlier in the simulation, there is another advantage to using probabilistic selection. This advantage is the ability to utilize multiple valid actions for a single need. Without probabilistic selection the agent would simply choose the action with the highest w_{PG} weight value and would in many cases not even learn other possible actions. However, with probability based goal selection and the changes made to w_{PG} weight calculations, the agent is able to learn multiple useful actions for a single need while still being able to differentiate the "quality" of the actions.

C. Changes to w_{pg} weight calculations

Here we show the effect of approach we took to w_{PG} weight calculation discussed in section II-D. In the example depicted in Fig 8, we see the w_{PG} values for a scenario with particular primitive pain possessing 3 useful solutions with derivative values $ds/d\hat{s}$ equal to 4, 2 and 1.5, respectively. Each solution in the scenario has a different utility, which is eventually translated to the maximum peak of each line in the figure.

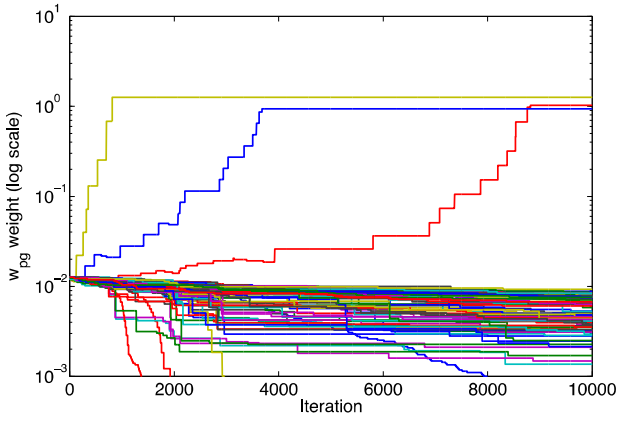


Fig. 8. Example of w_{PG} weights for a pain with multiple valid actions.

Note how in Fig. 8 useful actions separate from those that are not useful. Also, observe how the action with the highest value of $ds/d\hat{s}$ takes precedence over the other possible actions. Furthermore, because it has a higher utility, it increases sooner and at a higher rate due to the effect of (12). By the time the simulation ended, all three possible solutions were discovered and their weights reach their peak values. Furthermore, because of the probabilistic approach, which chooses action based on probability calculation derived from the w_{PG} weights, the potential solutions will have preference over other actions, with the most "useful" action still having the highest probability of being selected.

V. Comparision to Reinforcement Learning

In this section, we discuss comparing our proposed Motivated Learning (ML) algorithm with traditional reinforcement learning algorithms, including Q-learning, SARSA(λ) and hierarchical reinforcement learning. For fair comparison, we set the three reinforcement learning algorithms in the same environment and with the same environment responses to the agent's actions.. The settings of reinforcement learning algorithms are provided as follows:

Q-learning: Q-learning is one of the traditional reinforcement learning algorithms and can be used to acquire optimal control strategies from delayed rewards. The agent usually has no prior knowledge of the effects of its actions on the environment. Here, we implement the Q-learning algorithm from [15] with the discount factor $\gamma = 0.9$ and $\alpha = 0.7$.

SARSA(λ): The transitions from state-action pair to state-action pair are considered in this algorithm, which is a typical type of temporal difference (TD) learning algorithms. Sarsa(λ) is the combination of the Monte Carlo and dynamic programming ideas. It updates the estimates based on the other learned estimates, without waiting for a final outcome. We implement this Sarsa(λ) algorithm from [16] and set the parameters as $\gamma = 0.9$, $\alpha = 0.4$, and $\lambda = 0.9$.

Hierarchical RL: MAXQ is adopted to implement the hierarchical reinforcement learning from [17], where the subgoals and subtasks are defined. By doing this, the agent constructs a set of policies that need to be considered during reinforcement learning. The MAXQ value function is assumed

to represent the value function of any given hierarchy. The parameters are set as $\gamma = 0.9$, and $\alpha = 0.4$.

In order to compare these algorithms, we designed a "black box" environment that would present state and reward information to the RL or ML algorithm and receive a response in the form of an action to take. This action would then be presented to the "environment," which would adjust itself accordingly and respond with a reward value form 0-1 depending on the action and the current state of the environment. We show the results of the comparison of these mentioned algorithms in the next section. The environment itself is a simple "straight" 8-level hierarchy of "resources" similar in structure to the basic scenario presented in [13]. In this scenario there is a single "primitive" need, which can be resolved by the correct action, which consumes a specific resource. This specific resource gets depleted over time and needs yet another action/resource combination to restore itself, and so on up to the "top" level of the hierarchy, which is not depleted.

Reinforcement Learning Results

In Fig. 9 we compare our Motivated Learning algorithm against the reinforcement learning algorithms mentioned earlier in this section. Each algorithm was run 10 times and the results averaged to display in the figure. The 95% confidence interval was less than ±2.6% per line. As expected, HRL performs better than either Q-learning or SARSA. However, ML outperforms all of them. ML is simply better suited to operate in the hierarchically structured environment we provided.

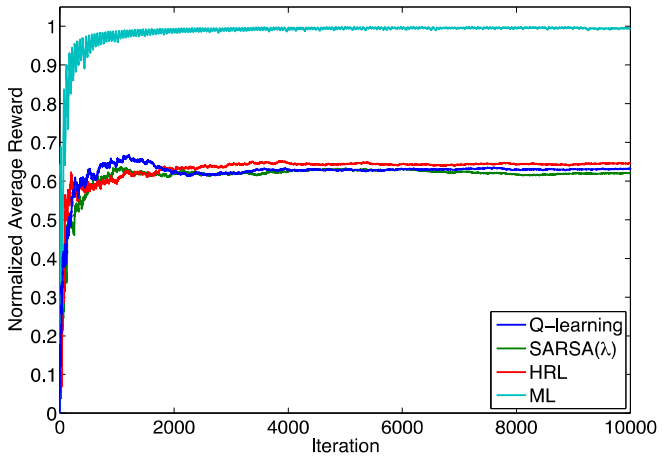


Fig. 9. Comparison of Reinforcement Learning algorithm average reward performance to Motivated Learning.

With these results we have shown that the Motivated Learning algorithm performs favorably against several common reinforcement learning algorithms. The results support our assertion that ML outperforms RL in complex environments, particularly, when an agent needs to discover the relations between several different "resources." RL is only using information about the environment state and a "reward" signal, while the ML agent's decision also depends on the internal state of the agent and its intrinsic rewards.

VI. Conclusion

We modified our Motivated Learning algorithm and discussed how these modifications affect the learning process. These modifications included calculation of need/pain biases, pain-goal weights, selection of actions, and setting the desired level or resources. We changed how we adjust weights in the agent and select goals that our agent is able to learn in a wide range of situations. The agent can now learn multiple useful actions for a single need, yielding a more flexible behavior. A detailed approach to setting the desired level of resources was also examined and implemented in our code. Additionally, we have performed tests using our ML agent and various RL algorithms within the same environment configuration to compare our ML algorithm against standard reinforcement learning algorithms.

References

[1] J. A. Starzyk, Motivated Learning for Computational Intelligence, in Computational Modeling and Simulation of Intellect: Current State and Future Perspectives, IGI Publishing, ch.11, pp. 265-292, 2011.

[2] Oudeyer, P-Y., Kaplan, F., & Hafner, V. V. (2007). Intrinsic motivation systems for autonomous mental development. IEEE Transactions on Evolutionary Computation, 11, 265–286.

[3] J. A. Starzyk and J. Graham "A Goal Creation System With Curiosity" 13th Int. Conf. on Cognitive and Neural Systems (ICCNS), Boston University, May 27-30, 2009.

[4] Pfeifer, R. & Bongard, J.C. (2007). How the Body Shapes the Way We Think: A New View of Intelligence, The MIT Press, 2007.

[5] K. Merrick, A Comparative Study of Value Systems for Self-Motivated Exploration and Learning by Robots, IEEE Transactions on Autonomous Mental Development, Vol 2(2):pp. 119-131, 2010.

[6] J. A. Starzyk, "Motivation in Embodied Intelligence," in Frontiers in Robotics, Automation and Control, I-Tech Education and Publishing, Oct. 2008, pp. 83-110.

[7] MacDorman, Karl F. (1999). Grounding symbols through sensorimotor integration. Journal of the Robotics Society of Japan, 17(1), 20-24.

[8] Harnad, S. (1990) The Symbol Grounding Problem. *Physica D* 42: 335-346.

[9] B. Masse, et al. How Is Meaning Grounded in Dictionary Definitions? *Int. Conf. Computational Linguistics, Coling 2008*, Manchester, 2008.

[10] Pezzulo G, Barsalou LW, Cangelosi A, Fischer MH, McRae K and Spivey MJ The mechanics of embodiment: a dialog on embodiment and computational modeling. *Front. Psychology, vol2, no.5, Jan. 2011.*

[11] J. Graham, J. A. Starzyk, D. Jachyra, "Opportunistic Motivated Learning Agents", 11th Int. Conf. on Artificial Intelligence and Soft Computing, Zakopane, Poland, Apr 29, May 3, 2012.

[12] J. Graham, J. A. Starzyk, and D. Jachyra, "Opportunistic Behavior in Motivated Learning Agents", submitted to IEEE Trans on Neural networks and Learning Systems, 2012.

[13] J. A. Starzyk, J. T. Graham, P. Raif, and A-H.Tan, "Motivated Learning for Autonomous Robots Development", Cognitive Science Research, v.14, no.1, 2012, p.10(16) pp. 10-25.

[14] J. A. Starzyk, J. Graham, L. Puzio, "Simulation of a Motivated Learning Agent," in Proc. AIAI 2013, pp. 205-214.

[15] T. M. Mitchell, "Machine Learning", New York, NY, USA: McGraw-Hill, 1997.

[16] R. S. Sutton and A. G. Barto, "Reinforcement Learning: An Introduction", Cambridge University Press, 1998.

[17] T. G. Dietterich, "Hierarchical Reinforcement Learning with the MAXQ Value Function Decomposition", Journal of Artificial Intelligence Research 13 (2000), 227-303.