Opportunistic Behavior in Motivated Learning Agents

James Graham, *Member, IEEE*, Janusz A. Starzyk, *Senior Member, IEEE*, and Daniel Jachyra, *Member, IEEE*

***Abstract*— This paper focuses on the novel motivated learning (ML) scheme and opportunistic behavior of an intelligent agent. It extends previously developed ML to opportunistic behavior in a multitask situation. Our paper describes the virtual world implementation of autonomous opportunistic agents learning in a dynamically changing environment, creating abstract goals, and taking advantage of arising opportunities to improve their performance. An opportunistic agent achieves better results than an agent based on ML only. It does so by minimizing the average value of all need signals rather than a dominating need. This paper applies to the design of autonomous embodied systems (robots) learning in real-time how to operate in a complex environment.**

***Index Terms*— Cognitive model, motivated learning (ML), opportunistic agent, reinforcement learning (RL).**

I. Introduction

An INTELLIGENT agent must have the ability to recognize new situations and important events through interactions with its environment. Therefore, the agent must be able to perceive and interpret external signals from the environment, which allows the agent to accurately and dynamically predict new situations, so that the agent can make decisions and act according to its objectives. The preferred approach is to have minimum supervision over the agent's actions and learning process, such that it can adapt to an unknown and changing environment that is quite often hostile to the agent.

Developing intelligent agents is important in a wide variety of applications, such as remote sensing, image recognition, quality control, warfare, assisting humans, entertainment, and so on. There are two fundamental questions regarding embodied agents: what motivates the agent to do anything and how to motivate the agent to explore the environment and interact with it in an innovative and effective way? It follows that determining a reliable motivation mechanism consistent with the agent's goals is an important issue.

Manuscript received June 22, 2013; revised May 18, 2014 and August 1, 2014; accepted August 27, 2014. Date of publication September 30, 2014; date of current version July 15, 2015. This work was supported by the National Science Centre, New Delhi, India, under Grant DEC-2011/03/B/ST7/02518.

J. Graham is with the Stocker Center, School of Electrical Engineering and Computer Science, Ohio University, Athens, OH 45701 USA (e-mail: jg193404@ohio.edu).

J. A. Starzyk is with the School of Electrical Engineering and Computer Science, Ohio University, Athens, OH 45701 USA, and also with the University of Information Technology and Management, Rzeszów 35225, Poland (e-mail: starzykj@gmail.com).

D. Jachyra is with the University of Information Technology and Management, Rzeszów 35225, Poland (e-mail: daniel.jachyra@gmail.com).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TNNLS.2014.2354400

There are several concepts for organizing motivational systems. One, introduced by Pfeifer and Bongard [1], shows motivation as a result of the developmental process. He believes that motivation for learning comes from the agent's growth and development. Another concept, based on external reward signals, is known in psychology as positive reinforcement. Its implementation led to a major development in machine learning known as reinforcement learning (RL) initialized by the work of Sutton and co-workers [2]–[4] followed by Brooks [5], Pfeifer and Scheier [6], Schmidhuber [7] and many others.

One of the main issues with RL is that it can suffer from the credit assignment problem [8]. To deal with the credit assignment problem in RL, a hierarchical reinforcement learning (HRL) algorithm as defined by Bakker and Schmidhuber [9] was developed. HRL works by using higher-level policies to determine useful subgoals and learning to apply them when appropriate. Although reinforcement desires to optimize the objective function (maximize the reward), it does not provide a mechanism for the development of motivations undefined by the designer. This was partially addressed introducing intrinsic motivations related to curiosity, or exploration of the most surprising events [10]. Schmidhuber [7] used the idea of intrinsic motivations to drive robots to explore and discover new subgoals. The intrinsic motivation system proposed by Oudeyer *et al.* [11] is similar to artificial curiosity presented by Bakker and Schmidhuber [9]. Oudeyer *et al.* [11] suggested an adaptive curiosity system where motivation comes from a desire to minimize prediction error. Generally, artificial curiosity helps to explore the environment and can be effective compared with purely RL-based exploration. Merrick [12] introduced motivated RL and compared several value systems for motivated exploration.

Steels [13] proposed a self-motivating agent with a flow mechanism. The main task was to give an agent the ability to self-regulate its own developmental process. An autotelic principle was proposed, which goes beyond classical RL as a behavioral method used by psychologists. He observed that when people perform a difficult task well that they are motivated to attempt even more complex tasks. The flow state is thought of as an optimal state of intrinsic motivation. The agent and its motivational mechanism are constantly trying to keep the balance between challenges and skills.

Merrick [12] pointed out that RL robots do not have internal drives to maintain their resources within the acceptable range [12]. To address this problem, a motivated learning (ML) system was proposed to allow the agent to develop its motivations and goals [14]. This goal-driven mechanism is based on primitive needs measured by pain signals that motivate the machine to act, learn, and develop. These mechanisms have been presented and discussed in [15]. Whereas standard RL relies primarily on external reinforcement provided by a teacher; ML relies on pain signals internal to the agent. Initially, these signals share some similarity with reinforcement-based reward signals, as they rely on predefined needs and guide the agent to learn specific skills. However, one of the main tenants of ML is the ability of the agent to derive additional motivations on the basis of its initial needs.

A machine operating based only on predefined needs would spend its entire existence trying to satisfy those needs and not generalize beyond them. Furthermore, it would be difficult for it to develop the understanding needed for more complex behaviors. Some might argue that more complex needs could be given to the agent, such as the need to drive a car. The problem with this approach is that it requires a plethora of background information and task-specific knowledge. Providing the agent with this need both defeats the purpose of developing a learning agent and curtails the agent's ability to discover and generalize knowledge.

Predefining knowledge and needs by itself is not bad; it is done extensively in nature. However, providing an agent with too much specialized knowledge can easily limit its development. The advantage of limited prior knowledge can be seen when comparing human development to that of many other animals. Human children are among the most limited at birth, and take well over a decade to develop into adults, but are also among the most versatile beings on the planet.

In [16], we demonstrated that in a dynamically changing environment, ML outperforms RL with no internal goal creation and can both maximize total reward and minimize the overall pain signals perceived by the agent.

The main contribution of this paper is to introduce opportunistic behavior to our ML agent. We demonstrate that such a modified agent can improve its performance by considering various motivations it may have at any given moment. In ML described in [15] and [16], the agent responded to its strongest need or pain signals and tried to minimize the strongest pain, if several of its needs were not satisfied. We show that by adopting opportunistic behavior the agent may reduce the average pain, and thus perform more successfully.

Therefore, one of the major objectives of this paper is the implementation and testing of opportunistic behavior in our ML agent. In our paper, we developed things such that the opportunistic agent is motivated to act by its internal motivations, creates abstract goals, and learns how to respond autonomously to the open-ended challenges and changes in its environment. The need for opportunistic behavior has been recognized in robotics, although it is defined differently than in our paper. For instance, a form of opportunistic behavior was considered in belief–desire–intention (BDI) agents [17] where the agent makes choices at run time rather than at the design stage. This facilitates the agent to operate in dynamic, open environments, whose behavior cannot be described at design time. Saffioti *et al.* [18] proposed using goodness, competence and conflict as categories for action selection criteria. Anselme [19] described opportunistic behavior in robot when it can identify favorable conditions for its actions. As it was pointed out in [19], autonomous robots are equipped with motivational states that determine when they will perform a specific task and that this does not allow the robot to recognize opportunities in the environment the way that animals do. He pointed out that the main issue for opportunistic behavior is to build an algorithm that allow the robot to converge toward a solution of its survival problems.

Kruse and Kirsch [20] defined opportunity as an alternative action to the robot's current action and choosing the next action was considered an opportunistic behavior. This approach was developed to facilitate robot interaction and cooperation with humans. However, opportunistic behavior in [20] was driven by human actions and did not result from the robot's own decision-making process. This was a supervised way of teaching a robot to take opportunities; thus, it is not appropriate for autonomous robots. Robots trained according to [20] would not know how to choose new opportunities, other than those a human taught it to use.

This paper is developed within the framework of the cognitive model proposed in earlier work, and briefly discussed in Section II along with basic properties of ML. Section III contains an overview of opportunistic motivated learning (OML). Two OML algorithms are presented and are related to the traveling salesmen problem (TSP). Section IV discusses computational efficiency and time requirements for the proposed algorithms. Section V presents two implementations of ML agents in virtual environments and show real-time simulation results for agent decision making and resulting total pain. Section V presents the conclusion.

II. Background

A. Characterization of ML Approach

Our prior work [16] demonstrated how motivated learning works in a dynamically changing environment and that it can outperform learning methods that neither generate nor reward internal goals. In this earlier work, the main focus was on the motivation mechanism, where only the core motivation building, goal creation, and symbolic sensory-motor I/O were used. Goal creation is different from goal inference [21] or subgoal generation as discussed in [22] and [23] where it is an adoption of existing goals rather than creation of new goals by the agent. Such limitations in an agent's ability to set the goals for itself would limit its autonomy and may compromise its performance.

The general motivation of our agent is to succeed in an unknown environment. Our view of motivations is in agreement with views of psychologists like Maslow [24]. However, unlike Maslow we do not predefine a hierarchy of needs, but try to evolve them through learning. Our approach, although based on predefined physiological needs, leads to higher level needs like safety, friendship, and so on. These needs are predefined by Maslov in his hierarchy of needs; however, the ML agent's hierarchy of needs evolves automatically as the agent learns.

To clarify our discussion, let us define some critical concepts used in ML.

Definitions: An agent has predefined *needs* (such as need for shelter, food, or energy level). A *primitive pain* is related to each predefined need and is defined as a measure that reflects how far the agent is from satisfying its need. The pain is larger, if the degree of satisfaction of a need is lower. For example, the following function can measure resource related pains:

$$P_i = w * \frac{R_d(s_i)}{\varepsilon + R_c(s_i)} \tag{1}$$

where R_d is a desired level of needed resource s_i, w is a weight that increases with the increased importance of resource i, R_c is the current level, and ε is a small positive number to prevent division by zero when $R_c = 0$. It was arbitrarily set to 10^{-10}. The agent acts on its need only if the pain is greater than a prespecified threshold. An agent's *motivations* are to satisfy its needs, which mean that the agent must reduce the associated pains below threshold. *Pain reduction* in ML is equivalent to a *reward* in RL.

When an agent is introduced to a new environment, it does not know how to satisfy its needs and must experiment with various resources and available actions. It attempts various sensory or motor actions in the environment until it succeeds in reducing its primitive need. In ML, the agent creates a new *abstract need* associated with the resource used to reduce the primitive need.

Once the agent learns that the resource is important and when the availability of the resource declines, the agent's need for the resource increases. The agent will pursue the reduction of the new abstract need in the same manner it sought the reduction of its primitive need.

Abstract pain is defined as a measure that reflects how far the agent is from satisfying its abstract need and is computed on the basis of the level of satisfaction of the abstract need in a similar way to (1).

In this context, the *performance metrics* that we use to compare various learning agent implementations is based on the sum of all pain signals that the agent experiences during its operation. The obvious goal is to minimize this sum.

Once introduced, an abstract need can be satisfied by acting on another resource. This leads to another higher level abstract need and related abstract pain. This simple mechanism allows the agent to build a potentially complex network of needs. We use the term network, because the needs can be interdependent on each other. For example, one resource, like money, could impact several needs and there are several ways with which one can acquire money.

This departs from classical reinforcement where the agent chooses its next action only on the basis of the state of the environment. In our version of ML, the agent chooses its action on the basis of the state of the environment and its current internal state. Major differences between our version of motivated learning and classical RL are shown in Table I and are briefly explained next.

The ML agent uses a separate value function for each of its motivations, so depending on the current motivation it may chose different actions for the same environment state.

TABLE I
REINFORCEMENT LEARNING VERSUS MOTIVATED LEARNING

Reinforcement learning	Motivated learning
Single value function	Multiple value functions
Measurable rewards	Externally immeasurable rewards
Predictable	Unpredictable
Objectives set by designer	Sets its own objectives
Maximizes the reward	Solves minimax problem
Potentially unstable	Always stable
Always active	Acts only when needed

Because rewards are internal to the ML agent, the total reward is not externally measurable, therefore the agent cannot be optimized and its behavior is, to a large degree, unpredictable, because an ML agent sets its own objectives on the basis of designer-controlled primitive needs and self-assigned abstract needs, whereas an RL agent's objectives are fully controlled by the designer and all the rewards are external and measurable. Thus, an RL agent can be optimized by the designer as described in [25], but our ML agent cannot, because the value functions are not externally measurable.

ML solves a minimax problem and, thus, is well endowed to deal with multiple needs. This is different from RL goals, which are based on maximization of reward. An ML agent is always stable, because its reward is always constrained, and only acts when needed as indicated by the need or pain level. Once all needs are below threshold, the agent does not have to act.

In much ML literature, motivations are fixed and drive either learning or the creation of new goals [11] or [26]. We see this as a limitation in developmental agents, similar to some extent to when the goals are explicitly given. The motivations to act cannot be *a priori* defined for an intelligent agent and they must evolve during its learning process. Yet, to be useful motivations must be grounded in the specific needs of the agent that are set by its designer. Thus, motivations in our approach to ML are derived from the needs of the agent and they result from the goal creation process [13], [14].

Opportunistic behavior is introduced within the framework of motivated learning. In ML, the agent interacts with the environment making real-time decisions on what to do. It combines the creation of goals and needs according to the state of the environment and its internal state. Algorithm 1 implements the agent operations.

Notice that the ML algorithm works in real-time and the agent interprets inputs from the environment at each iteration. Thus, a goal may be changed if the conditions in the environment change.

As shown in [16], an ML agent performs better than the TD-Falcon RL agent in a dynamically changing environment in which the amount of resources needed by the agent depends on the agent's action.

It is true that not all environments are as difficult and dynamic as this one, but because there was no situation tested in which RL without internal goals outperformed ML and we could demonstrate the opposite, we consider this as evidence of the superior performance of ML over traditional RL.

Algorithm 1 Motivated Learning Algorithm

```
1. The agent reads the current state of the environment.
2. With the sensory inputs and its internal state, the agent
   evaluates the internal pains.
3. The agent checks whether the pains were reduced.
   a. If a pain was reduced, the agent learns proper
      behavior by using classical RL and creates or
      reinforces an abstract need.
   b. Otherwise, it learns that the action is not useful
      and reduces the chance of repeating it in a similar
      state of the environment and the agent.
4. The pains are updated using the degree of satisfaction
   of its needs (1). If a new need was established due to a
   successful action, the related pain is evaluated as well.
5. Select a goal according to the existing pains and the
   state of the environment.
6. Perform a desired action to accomplish the goal.
7. Repeat 1–6.
```

B. Comparison to Other Methods

The ML algorithm shares some of the same elements of BDI agents [27]–[30]. For example, an ML agent has belief that its internal representation of the state of the world around represents the state of the said world. The internal representation or belief of the ML agent is formed through the system of perceptions, pain centers, goals, and planned actions, and links its perception of the environment to semantic knowledge about the environment. Desires in a BDI agent correspond to motivations as expressed by the pain centers in an ML agent. The pains compete for attention and drive the ML agent's actions. Intentions are similarly represented in an ML agent as the selected method for implementing a goal chosen by the agent. BDI and ML agents differ primarily by the way their actions are determined. ML agents can generate their complex abstract motivations (desires) and derive complex actions from them, whereas BDI agents typically have their actions determined by their designers, and lack the ability to define desires beyond their initial design.

C. Opportunistic Behavior

In ML, an agent simply pursues its most pressing needs. Selection of a goal is obtained using a winner that takes all approach. The opportunistic motivated agent considers all its needs and the efforts required to accomplish them. There will be instances where a less pressing need can be accomplished easily, yielding lower time and effort to reduce the average of all pain signals. For instance, this happens when the agent must travel a long distance before it can work on its most pressing need, whereas another need can be easily satisfied without such extra effort. With this opportunity the agent will lower its average pain. Such behavior is in agreement with the behavior of animals.

In animals, the stronger their motivation to act, the more focused they are on a specific goal and less distracted by other potential opportunities [19]. This kind of behavior is typified by the ML mechanism we presented in [14] and [15]. However, if the dominant motivation is not so strong, an animal may choose to pursue a less pressing need when an opportunity arises. We would like to model a similar behavior in the embodied agents' control. In [31], we presented the general idea of an opportunistic agent, where the agent decided what to do was based on the winning action value.

Definition: By opportunistic behavior, we mean actions of the agent reducing not necessarily the most pressing need, but the one leading to the largest reduction of the average overall pain.

For example, let us examine a situation where an individual is on his way to the bank to cash a check. While driving to the bank, he passes by a supermarket and realizes that he needs groceries. He decides to stop and buy groceries before continuing on, thus reducing a (less critical) need for food. In performing opportunistic actions, the agent has to balance several factors, such as, current need levels, distances involved, time needed to perform various actions, and predicted (or known) changes in pains or needs. In opportunistic behavior, the agent has to continuously reevaluate its options, because its pain levels change dynamically both as a result of its actions and changes within the environment.

Section III provides a mathematical treatment of this opportunistic behavior reducing it to an optimization problem with (7) and (8) being different instantiations of the optimized functions.

In a standard implementation of ML, the agent selects a goal in step 5 of the ML algorithm choosing the maximum pains that can be resolved considering current conditions in the environment. This is simply obtained using winner takes all competition between pain signals with inhibition to those goals which cannot be currently performed. Although such implementation of ML works very well, this can be improved using the opportunistic behavior developed in this paper.

The opportunistic ML agent implements step 5 of the ML algorithm as follows.

1) Evaluate the effort to reduce each pain according to the resource location and the required task time.
2) Use a heuristic algorithm to select the next action considering not only the pain level but also the cost of performing the action (doing this requires that the agent evaluates total cost of satisfying all its needs).

The main focus point of this paper is the choice, organization, and properties of the heuristic algorithm that the ML agent uses in step 5 of the ML algorithm. These are discussed in Sections III and IV.

Although pains (and goals) may change in over time, the agent makes its opportunistic decision according to the current state of the environment and its needs. As conditions of the environment changes, the agent reevaluates its previous choices dynamically adjusting to changing environment and its internal needs.

In humans, opportunistic behavior involves attention switching to evaluate various opportunities in a changing environment. We adopted a similar approach to design cognitive agents. Fig. 1 shows how attention switching fits into our cognitive model described in [32]. Implementing this cognitive model is our long-term objective.

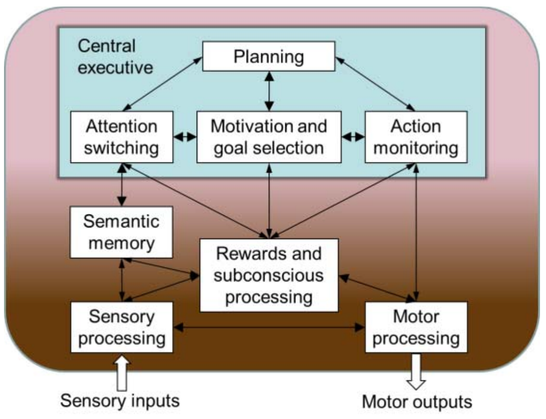

Fig. 1. Cognitive model pursued in this paper.

III. Opportunistic Agent

The opportunistic motivated agent (OML) considers all pains (needs) and the total effort to reduce them, trying to minimize its cumulative pain. The method we chose and the reasons for the choice will be discussed in this section. It is based on our need to limit computation time while improving overall pain reduction.

Opportunistic behavior is based on the interplay of motivations that the agent develops while learning proper behavior. This is based on our prior work on ML. Using ML, the agent develops needs and related pains that must be satisfied at any given moment. Opportunistic behavior is used to tell the agent which of these needs should be considered first by prioritizing them as we discuss in this section.

Let us consider multiple pains that the agent can reduce by taking proper actions. Assume that the agent must spend some time performing each action to reduce a pain and additional time to travel to the proper destination to perform these actions. Also assume that the agent does not anticipate future pains, so it can minimize the expected cumulative pain according to the current state of various pains. Finally, assume that the agent knows the time required to travel between different destinations and the effort required to finish each job. Dynamic changes that take place in the environment may affect these travel times and the efforts, but we assume that the agent knows their current values.

To find the minimum cumulative pain the agent chooses the travel route and estimates the cumulative pain along this route. This cumulative pain P_c can be estimated from

$$P_c = \min_r \left\{ \sum_{k=1}^{n} p_k * \left(\sum_{i=1}^{k} \left(t_i^{\text{tr}} + t_i^{w} \right) \right) \right\} \tag{2}$$

or as expressed in an equivalent form

$$P_c = \min_r \left\{ \sum_{k=1}^{n} \left(\left(t_k^{\text{tr}} + t_k^{w} \right) * \sum_{i=k}^{n} p_i \right) \right\} \tag{3}$$

where r is the traveled route between various destinations that the agent selects, n is the number of destinations, t_k^{tr} is the time to travel from point k to $k+1$ along the route, t_k^{w} is the working time to complete a task at the kth destination along the route, and p_i is the pain at point $i, i = 1, \ldots, k$. Notice that at the beginning of the route ($t_1^{w} = 0$), the agent may not be at the job location, so in such a case the corresponding $p_1 = 0$. Also notice that each individual pain effect is cumulative over the whole task until the pain is reduced or eliminated. This is particularly obvious from (3) where each individual part of the route that ends with performing a task at location k is multiplied by the sum of the pains in all remaining locations not yet visited by the agent.

Equations (2) and (3) estimate the cumulative pain that the opportunistic agent will experience assuming that pain is known ahead of time and is fixed. We can generalize this result and assume that pain at each location changes over time. In such case, we can modify (3) as follows:

$$\begin{aligned} P_{\text{av}} &= \min_r \left\{ \sum_{k=1}^{n} \left(\left(t_k^{\text{tr}} + t_k^{w} \right) * \sum_{i=k}^{n} \left(\frac{1}{t_k^{\text{tr}} + t_k^{w}} \int_0^{t_k^{\text{tr}} + t_k^{w}} p_i \, dt \right) \right) \right\} \\ &= \min_r \left\{ \sum_{k=1}^{n} \left(\left(t_k^{\text{tr}} + t_k^{w} \right) * \sum_{i=k}^{n} P_{i\,\text{av}} \right) \right\} \end{aligned} \tag{4}$$

where $P_{i\,\text{av}}$ is the average pain level at location $i, i = 1, \ldots, k$ over the duration of time to travel kth section of the route and complete kth task.

In a similar way, we can generalize (2) as follows:

$$P_c = \min_r \left\{ \sum_{k=1}^{n} P_{k\text{av}}^{c} * \left(\sum_{i=1}^{k} \left(t_i^{\text{tr}} + t_i^{w} \right) \right) \right\} \tag{5}$$

where $P_{k\text{av}}^{c}$ is the average pain level at location k over the duration of time to travel all sections from section 1 to k of the route and complete all corresponding tasks along these sections. Although (2) and (3) are equivalent, (4) is easier to compute than (5) because $P_{k\text{av}}^{c}$ depends on what was the pain change over previously traveled sections, so it cannot be precomputed even if these changes are anticipated and can be predicted over any specific time interval.

Lemma: Finding the minimum P_c is an nondeterministic polynomial time (NP) complete problem.

Proof: We can prove it by reducing this problem to TSP, which is known to be NP complete. The simplest case of the cumulative pain minimization is when the agent is at a specific target destination and plans its traveling route and all $t_k^{w} = 0$. In addition, let us assume that all p_i are constant and $p_i = 1/n$. In such a case, (3) is reduced to

$$P_{\text{av}} = \min_r \sum_{k=1}^{n} t_k^{\text{tr}} w_k \tag{6}$$

where $w_k = 1$. TSP is a special case of (6) with all $w_k = 1$ and t_k^{tr} replaced by the distance between points k and $k+1$. $\square$

Because finding the optimum solution to the opportunistic behavior problem is NP complete, we can use heuristics to approximate the minimum. Although many heuristic algorithms exist for TSP, none of them can be directly used in this problem because weights w_k in (6) depend on the path selected.

Next, we present two heuristic algorithms that the opportunistic agent may use to control its behavior.

Algorithm 2 Linear Heuristic Algorithm

```
1. Iterate with $i = 1, \dots, n$, where $n$ is the number of pain locations that the agent considers. Set the initial location as the current location of the agent, and set $m = n$, and $d_{\mathrm{LH}} = \emptyset$ where LH is linear heuristic.
2. At the current location of the agent, which is equal to pain reduction rate (7), find the maximum where $t_k^{\mathrm{tr}}$ is a distance from the agent at the current location to the pain at location $k$.
3. Append node $d_r$ to the selected path $d_{\mathrm{LH}} = \{d_{\mathrm{LH}}, d_r\}$ where $r$ corresponds to $k$ with the minimum value $P_{\mathrm{rr}}$ in (7).
4. Change the current location of the agent to the new location $r$, remove location $r$ from pain locations, and repeat steps 2 and 3.
5. The resulting optimized route is $d_{\mathrm{LH}} = \{d_1, d_2, \dots, d_n\}$.
```

TABLE II
LOCATIONS AND PAIN SIGNAL VALUES

Location Number	1	2	3	4	5	6	7
x Coordinate	2	2	15	8	12	18	12
y Coordinate	2	19	8	14	9	2	18
Pain Level	10	15	7	30	15	16	20

A. Linear Heuristic (LH) Algorithm: Pain Reduction Rate

One of the simplest heuristics for TSP is a greedy strategy in which the agent goes to the closest location. We can adopt a similar strategy in the opportunistic agent, but instead of selecting the nearest location, the agent selects the location where it can get the largest pain reduction with the smallest effort. In this algorithm, the agent determines the pain reduction rate defined as

$$P_{\mathrm{rr}} = \max_r \left(\frac{p_k}{\left(t_k^{\mathrm{tr}} + t_k^{w}\right)^2} \right), \quad k = 1, \dots, m \tag{7}$$

where k is one of the possible locations reachable from the current position of the agent.

The LH algorithm for the opportunistic agent is explained in Algorithm 2.

This algorithm is simple to implement, has linear complexity, and, thus, is useful for a complex environment where the agent faces many choices and needs to update them in real-time. Moreover, in spite of its simplicity the algorithm delivers near optimum performance in tests.

Example 1: Assume that an agent is placed on a 2-D grid and the starting location of the agent is $(xy) = (10, 7)$. Assume that at current iteration of the ML algorithm, the agents determined its needs, related pains, and observed the locations of the resources it can use to satisfy its needs. Assume that there are seven resource location points with x and y coordinates and the pain levels to be reduced at these points as shown in Table II.

Distances between different locations are as shown in Table III.

For the given example, the pain reduction rates computed using (7) are as shown in Table IV.

TABLE III
DISTANCES BETWEEN VARIOUS LOCATIONS THAT OPPORTUNISTIC AGENT NEEDS TO VISIT

Distance	1	2	3	4	5	6	7
1	0.00	17.00	14.32	13.42	29.73	17.46	16.40
2	17.00	0.00	17.03	7.81	14.87	18.87	10.77
3	14.32	17.03	0.00	9.22	22.20	3.16	7.62
4	13.42	7.81	9.22	0.00	16.49	11.18	4.12
5	29.73	14.87	22.20	16.49	0.00	21.84	15.00
6	17.46	18.87	3.16	11.18	21.84	0.00	8.49
7	16.40	10.77	7.62	4.12	15.00	8.49	0.00

TABLE IV
COMPUTED PAIN REDUCTION RATES

Location	1	2	3	4	5	6	7
P_{rr}	0.96	0.97	1.15	3.62	3.92	1.53	1.64

Following the LH algorithm, the opportunistic agent selects to go to location 5—its nearest location. Notice that standard ML agent would go to location 4 with the largest pain (as shown in the last row of Table II). Here, we assume that t_k^{tr} can be replaced by the traveling distances shown in Table III, and that all values of t_k^{w} are equal to 1. Using this heuristic strategy, the opportunistic agent will chose the route through the following nodes: $d_{\mathrm{LH}} = \{5, 4, 7, 2, 1, 6, 3\}$, while the route of the ML agent will have nodes: $d_{\mathrm{ML}} = \{4, 7, 6, 5, 2, 1, 3\}$. This will result in a 25.9% reduction in the total pain of the opportunistic agent over the pain suffered by the ML agent. The work time in this experiment was set to 1 for all tasks.

Obviously, the advantage of opportunistic behavior over ML is greater if the agent would be forced to go back and forth chasing the biggest pain rather than taking the opportunity offered by nearby pain reduction.

Notice that the agent reevaluates its decision after each iteration of the ML algorithm, so it is likely that it will not complete the entire path. Nevertheless, the advantage of selecting the goal to be implemented according to the LH algorithm can be demonstrated in many simulated environments.

B. Quadratic Heuristic (QH) Algorithm

The main idea behind the QH algorithm is to be able to paste together the entire path planned by the agent from the locally dominating section of the path. In this algorithm, the agents finds the minimum normalized effort reduction in the entire table

$$P_{\mathrm{rr}} = \min_{\mathrm{ik}} \left(\frac{\left(t_{\mathrm{ik}}^{\mathrm{tr}} + t_{\mathrm{ik}}^{w}\right)^4}{p_k} \right). \tag{8}$$

Locally the approach is similar to the greedy algorithm; however, because the search is along any possible path segments, rather than from the current agent location the agent has a chance to find a better total path that minimizes its cumulative pain (5).

The QH algorithm for an opportunistic agent is explained in Algorithm 3.

Example 2: To illustrate the steps of transforming the QH algorithm to a TSP problem, let us consider 1-D case in which the agent is located at coordinate $x = 5$, with resource

Algorithm 3 Quadratic Heuristic Algorithm

1. Compute combined effort matrix $(n+1) \times nM$ weighted with pain at each destination location

$$M = \begin{bmatrix} (a_1 + t_1^w)^4 & (a_2 + t_2^w)^4 & \dots & (a_n + t_n^w)^4 \\ (d_{11} + t_1^w)^4 & (d_{12} + t_2^w)^4 & \dots & \\ (d_{1n} + t_n^w)^4 & & & \\ \vdots & \vdots & \ddots & \vdots \\ (d_{n1} + t_1^w)^4 & (d_{n2} + t_2^w)^4 & \dots & (d_{nn} + t_n^w)^4 \end{bmatrix} * \begin{bmatrix} 1/p_{11} & 0 & \dots & 0 \\ 0 & 1/p_{22} & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 1/p_{nn} \end{bmatrix} \tag{9}$$

   where a_j is agent distance, t_j^w is the working time at the jth location, d_{ij} is distances between pain locations, and p_{jj} is the pain value at location j.

2. Create square matrix W from matrix M by adding a new column with ∞ values and change diagonal elements to ∞

$$W_{(\mathrm{n+1})\mathrm{x}(\mathrm{n+1})} = \begin{bmatrix} \infty & m_{11} & m_{12} & \cdots & m_{1n} \\ \infty & \infty & m_{22} & \cdots & m_{2n} \\ \infty & m_{31} & \infty & \cdots & m_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \infty & m_{n+1,1} & m_{n+1,2} & \cdots & \infty \end{bmatrix}. \tag{10}$$

   The ∞ value represents connections, which will never be chosen, as the agent must start from its current location and cannot revisit any location.

3. Normalize matrix W by dividing each element w_{jk} by the sum of finite elements in row j and column k of matrix W. In addition, change all ∞ values to 1 to obtain the normalized effort matrix V

$$V = \begin{bmatrix} 1 & v_{12} & v_{13} & \cdots & v_{1,n+1} \\ 1 & 1 & v_{23} & \cdots & v_{2,n+1} \\ 1 & v_{32} & 1 & \cdots & v_{3,n+1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & v_{n+1,1} & v_{n+1,2} & \cdots & 1 \end{bmatrix} \tag{11}$$

   where

$$v_{jk} = \left[\frac{w_{jk}}{\sum_{i \neq j}^{n+1} w_{ji} + \sum_{i \neq k}^{n+1} w_{\mathrm{ik}}} \right] < 1, \quad j = 1, \dots, n+1, \quad k = 2, \dots, n+1, \quad j \neq k.$$

4. Iterate with $i = 1, \dots, n$
   a. Find the smallest element of matrix V and store the row index of this element in the resulting rows vector R (setting $r_i = b - 1$), and columns vector C (setting $c_i = e - 1$).
   b. Replace all elements of row b and column e by 1. Because all elements of the matrix V are not larger than 1, this will eliminate possibility of selecting this element again.
5. To obtain the selected path, iterate with $m = 1, \dots, n$
   5.1 Start with $r_i = 0$, set $m = 1$, and use $b = c_i$ as the first visited location ($d_m = b$).
   5.2 Next, find $r_k = d_m$, increase m by 1, and use $b = c_k$ as the next visited location ($d_m = b$).
6. The resulting route is $d_{\mathrm{QH}} = \{d_1, d_2, \dots, d_n\}$.

TABLE V
PAINS AT VARIOUS LOCATIONS

Location number	1	2	3	4	5	6
x coordinate	2	10	20	38	80	93
Pain level	30	18	20	7	31	19

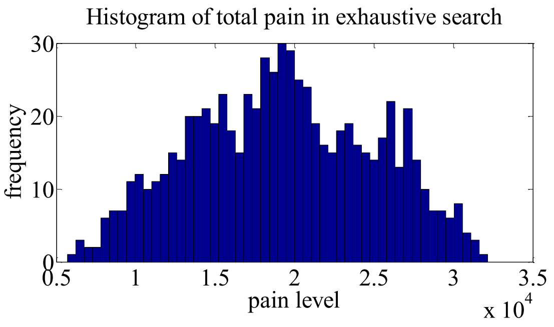

Fig. 2. Histogram of all solutions by using an exhaustive search.

locations (x coordinates) and the pain levels that can be reduced at these locations as shown in Table V.

First, we formulate the effort matrix M (9) and reduce this matrix to the normalized form V (11). Matrix M is equal to (only three significant digits are shown)

$$M = 10^4 * \begin{bmatrix} 0.00 & 0.01 & 0.33 & 19.1 & 108 & 330 \\ 0 & 0.04 & 0.65 & 26.8 & 126 & 377 \\ 0.02 & 0 & 0.07 & 10.1 & 82 & 262 \\ 0.43 & 0.08 & 0 & 1.86 & 44.7 & 158 \\ 6.25 & 3.93 & 0.65 & 0 & 11.0 & 51.8 \\ 130 & 141 & 69.2 & 48.8 & 0 & 0.20 \\ 239 & 27.7 & 150 & 140 & 0.124 & 0 \end{bmatrix} \tag{12}$$

and the normalized V matrix is

$$V = \begin{bmatrix} 1 & 1\mathrm{E}-06 & 8\mathrm{E}-06 & 5\mathrm{E}-04 & 0.027 & 0.130 & 0.202 \\ 1 & 1 & 3\mathrm{E}-05 & 9\mathrm{E}-04 & 0.034 & 0.139 & 0.221 \\ 1 & 3\mathrm{E}-05 & 1 & 1\mathrm{E}-04 & 0.017 & 0.113 & 0.171 \\ 1 & 0.001 & 1\mathrm{E}-04 & 1 & 0.004 & 0.078 & 0.114 \\ 1 & 0.014 & 0.008 & 0.002 & 1 & 0.025 & 0.041 \\ 1 & 0.170 & 0.174 & 0.114 & 0.077 & 1 & 1\mathrm{E}-04 \\ 1 & 0.202 & 0.225 & 0.146 & 0.133 & 1\mathrm{E}-04 & 1 \end{bmatrix}. \tag{13}$$

Using exhaustive search for this example, we can obtain all solutions with the histogram shown in Fig. 2 with a mean pain value of 19 229 and standard deviation of 5728. The minimum total pain was 5701, and the maximum was 32 159.

IV. COMPUTATIONAL EFFICIENCY

To compare the efficiency of the proposed solution with opportunistic agent behavior, we transform the description of

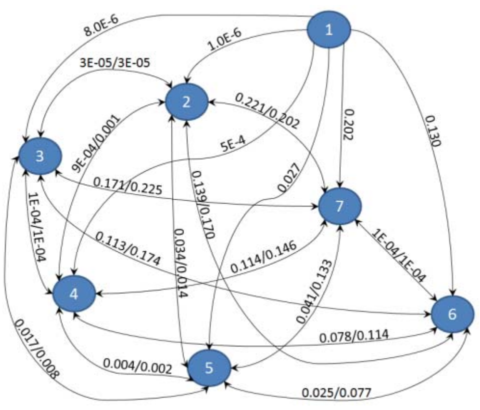

Fig. 3. Directed graph with seven nodes and 36 edges that correspond to values in matrix V (13). To simplify the drawing we used bidirectional edges.

TABLE VI
COMPARISON TOTAL PAIN TO CHRISTOFIDES LIMITS

Number of Nodes	Christofides Limit [35]	LH	QH
12	$L_a \leq 18389.8$	12981 (±1680.3)	16118 (±3812.5)
11	$L_a \leq 14964.6$	10689 (±1067.2)	13519 (±3135.2)
10	$L_a \leq 14501.2$	10358 (±1470.3)	11376 (±1674.8)
9	$L_a \leq 11655.1$	8476.4 (±1044.7)	9144.8 (±1287.6)
8	$L_a \leq 10308.1$	7496.8 (± 1241.5)	7946.1 (±1933.3)
7	$L_a \leq 9147.9$	6860.9 (± 708.7)	7498.9 (±1080.8)
6	$L_a \leq 7438.8$	5719.1 (±795.5)	5845.5 (±1034.6)

the problem to a format compatible with the asymmetric TSP described by a directed graph. Because TSP is one of the most researched problems in discrete optimization and has a large number of heuristics solutions, we can compare both efficiency and computational cost of the QH algorithm.

This procedure applied to the problem described in Example 2 resulted in the directed graph shown in Fig. 3. The matrix V from (13) and graph shown in Fig. 3 are used for the standard TSP. According to QH algorithm use a combined distance matrix with average pain at each location included in the graph through calculation of matrix M (9). Another important point is that our agent always starts from node 1, visits every node once, and does not go back to the starting point.

Christofides [33] showed a polynomial time algorithm for TSP such that the total length obtained by this algorithm L_a is less than $3/2 * L_{\min}$ where $L_{\min}$ is the global minimum that can be found using a heuristic process. In our case, this means that a polynomial time algorithm can find a solution with the total accumulated pain signal less than the Christofides limit $L_a < 3/2 * 5701 = 8551.5$.

In addition, Cornuejols and Nemhauser [34] demonstrated a tighter bound for Christofides' algorithm $L_a \leq (3m - 1/2m)L_{\min}$ where m is the largest integer not greater than $n/2$ and that the bound is tight (reaching equality) for $n > 6$. In our case, with $n = 6$, $m = 3$ Christofides algorithm gives $L_a = (3 * 3 - 1/2 * 3)5701 = 7601.32$.

As we can see from Table VI, both algorithms produce results below the limits of algorithmic solutions as specified by Christofides. By running the QH algorithm for Example 2,

TABLE VII
COMPARISON OF OPPORTUNISTIC AGENT WITH SEVERAL TSP ALGORITHMS

Method	Minimum total pain	Std, Dev.	Run time in seconds	Total pain for nodes 4-12
LH Algorithm	5701	1144	0.000733	68286
QH Algorithm	6519	1994	0.007729	68971
Simulated Annealing	12425	1289	0.076700	164414
Greedy Search	13159	455	0.024973	106429
Exhaustive Search	5701	1479	0.060810	62102
Ant Colony Algorithm	8624	626	0.350591	83343
Random Search	12827	1359	0.917594	168471
Genetic Algorithm	5701	95	5.372666	62102

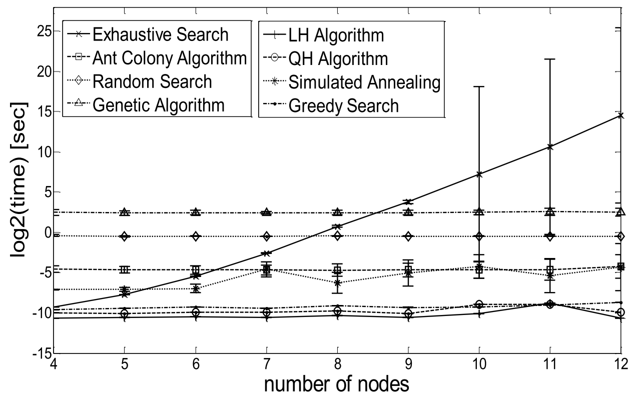

Fig. 4. Run time for different algorithms (with std-dev).

we got the route visiting nodes $d_{\text{QH}} = \{1, 2, 3, 4, 6, 5\}$ and total pain of 6519 and the opportunistic agent was better than 99.58% of all cases. The LH algorithm was even better in this case with total pain equal to 5701.

According to the LH algorithm, the opportunistic agent will chose the route visiting nodes $d_{\text{LH}} = \{1, 2, 3, 4, 5, 6\}$, whereas the ML agent will visit nodes in the order of decreasing pain signal values $d_{\text{ML}} = \{5, 1, 3, 6, 2, 4\}$ and will suffer 4.15 times larger cumulative pain than the opportunistic agent. Total accumulated pain for the opportunistic agent was 5701, while for the ML agent it was 23 701. The opportunistic agent was better than 99.86% of all cases, while ML was better than only 24.58% of cases.

We have tested the LH and QH algorithms against Christofides' bounds on a number of graphs starting from the graph shown in Fig. 3 and gradually increasing the problem complexity by adding more nodes (pain centers). For each size of the graph, we repeated the test randomly generating the graph structure. The average results with standard deviations obtained after 20 runs of algorithms are shown in Table VI.

We used the approach presented in Example 2 to formulate an equivalent TSP problem for opportunistic agent behavior and applied several popular TSP algorithms comparing both algorithmic complexity and quality of the final results. The results of analysis are shown in Table VII and Fig. 4 in ascending order of run time.

Fig. 4 shows the logarithm of the run time of the compared algorithms as a function of the number of nodes in the resulting directed graph.

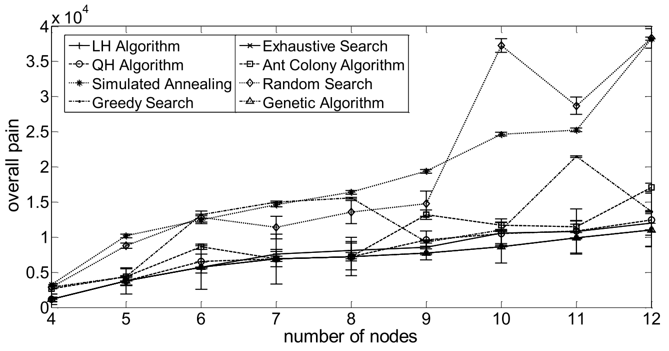

Fig. 5. Overall pain for different algorithms (with std-dev).

As can be seen from the presented results, the linear and quadratic heuristic algorithms are both fast and effective. Notice that although, for small graphs, the exhaustive algorithm is more efficient than ant colony, random search, or genetic algorithms, it is the most computationally expensive once the graphs have more than nine nodes. All other algorithms maintain their run-time level for tested size of graphs.

As shown in Fig. 4, the linear and quadratic algorithms require the least simulation time, whereas the exhaustive search is the most expensive one with an exponentially growing time requirement.

Fig. 5 shows the overall pain of the compared algorithms as a function of the number of nodes. Each data point shown in Fig. 5 is the average result obtained after 20 runs. Results demonstrate that both the linear and quadratic algorithms were almost as efficient as the genetic and exhaustive search algorithms that yielded the best results.

Because there is a large variability of the overall pain in various algorithms, we summarized total pain in all experiments with the number of nodes changing from 4 to 12 and presented this in the last column of Table VII. This confirms that the proposed algorithms are almost as efficient as the most computationally demanding ones, and are superior to simulated annealing, greedy search, ant colony, and random search algorithms in both the simulation time and total pain obtained.

In a typical situation observed in a simulated environments, an opportunistic agent must choose from 3 to 7 options (represented by the number of active pains), therefore we can recommend using the LH algorithm to control the behavior of the opportunistic agent. The quadratic algorithm has been useful to formulate opportunistic behavior in terms of the well-studied TSP and helped to make our final recommendation regarding the use of the LH algorithm.

The examples considered represent what an opportunistic agent may encounter while dealing with its pains. An agent may have only a small number of pains that are relevant at any given moment. Thus, the algorithm selection is specific to ML needs and corresponds to a small number of choices a cognitive agent considers in its working memory. We do not claim that these algorithms are effective to solve large-scale TSP problems.

To summarize, the opportunistic agent uses the lowest cost path, to improve every step of the ML process. The learning mechanism still uses abstract goal creation and internal motivations as described in our previous works [14]–[16].

Notice that the heuristic algorithms discussed in this section are to help the agent decide what to do to cope with the set of needs that it developed during its learning process. This should not be confused with the learning process itself. The agent learns what is good for it by interacting with complex dynamically changing environment and observing results of such actions, but the planning process it uses deciding what to do involves solving problems similar to TSP using one of the presented heuristic algorithms. So while TSP solves a search problem only, the action taken by the agent after the search is evaluated and is a basis for learning within ML approach. Specifically, actions that resulted in pain reduction will be reinforced. In addition, they may lead to change agent's motivations and result in new goals.

V. Simulating Agents

The implemented learning agent needs to be tested in real-time application. This can be done either via robotics, where the agent controls a robotic body, or via simulation, where the agent exists within a virtual environment and possesses a virtual body. These approaches require an embodied agent central to Embodied Cognition, initiated in the early-1990s with work by Brooks [5] and Pfeifer and Scheier [6]. The theory behind embodied cognition is that machine's interaction with its environment is predetermined by its embodiment and that intelligence cannot develop without embodiment.

This approach adheres to the idea that the agent's body shapes its development, because the motor and sensory apparatuses of the body effect how the agent functions.

Robotics platforms, can be costly to design, build, and operate; advanced robots cost upward of \$100 000. Furthermore, it is still necessary to create an environment for the robot to operate within. In a simulation, there is much more freedom in designing the agent and the environment, more freedom to change things, and physical hardware costs are limited to the costs of computers. Simulation and virtual environments were used to develop principles of synthetic intelligence agent in [35], use learning agents as nonplayer characters in computer games [36], and build robotic training platforms, [37], [38]. Thus, we chose to use simulated virtual agents to demonstrate their improved learning capabilities when the agent adopts an opportunistic behavior.

In our previous work, we implemented a basic ML algorithm in MATLAB. Results from some of the initial tests can be found in [14] and [16]. However, the environment and consequently the embodiment in these simulations were simple, limiting the agent's capabilities. Alternative environments are needed to expand the agent's capabilities, properly investigate various scenarios, and improve overall fidelity. We have tested the ML agent [39] in the NeoAxis game engine software development kit [40] by implementing the motivated agent functionality in C++.

OML implementation of the learning agent combines ML (providing pain signals and motivations) with a heuristic algorithm (to choose the current goal).

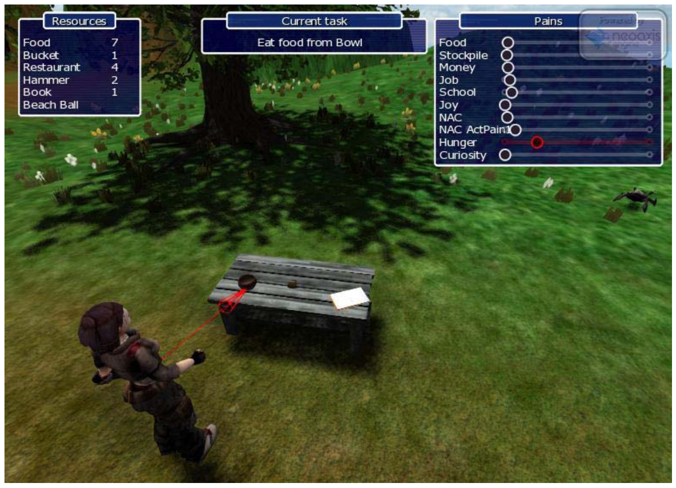

Fig. 6. Agent walking toward a target resource.

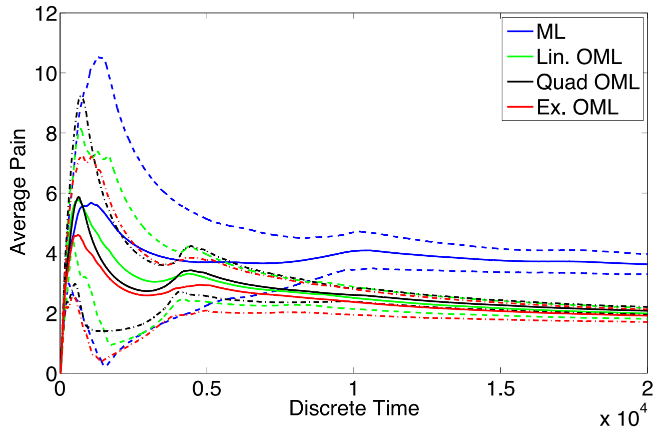

Fig. 7. Total average pain comparison for standard ML, linear OML, quadratic OML, and exhaustive ML algorithms.

In this section, we compare three implementations of the ML algorithm. The first one denoted here as ML is a standard implementation of ML where the action is chosen on the basis of the maximum pain signal, the second one, denoted by linear OML, is an opportunistic ML where the choice of action is according to the LH algorithm, and the third one, denoted by exhaustive OML, is an opportunistic ML where the choice of action is according to an exhaustive search.

The performance metric in this comparison is defined as the minimum total average pain suffered by each agent that is subjected to the same environment. The smaller the pain, the better the agent's performance.

A. NeoAxis

NeoAxis comes with a number of assets and demonstrations as well as Map and Resource Editors. The presence of several terrain maps eliminates much of the work needed to develop a realistic environment.

Fig. 6 shows a third person view of the opportunistic agent implemented in NeoAxis. It shows the agent about to act on a resource object in front of itself.

The depicted opportunistic agent, detects a useful resource within its view range, evaluates it against its needs and decides whether to deviate from its current objective or not. In this case, the agent's choice is determined by the LH algorithm discussed in Section III.

B. Opportunistic Simulation Results

This section provides the results of real-time comparison between the performances of the standard ML algorithm and the OML algorithm. The LH algorithm was used for the OML agent. Although this algorithm is not as effective as the exhaustive algorithm, it still provides noticeable benefits.

We used a simple farming scenario, where the agent has to farm for food and keep itself fed. There are five primitive pains in the scenario that are linked predefined needs of the agent. The agent can satisfy its primitive needs by using various resources and create up to eight abstract needs.

Fig. 7 shows a comparison of total average pains for four algorithms (ML, linear OML, quadratic OML, and exhaustive OML) by using 20 000 iterations for each algorithm in a simulated scenario. The total average pain is the cumulative sum of all pains, divided by the iteration number, and is used as the performance measure of the particular run. The agent's objective is to lower this total average pain.

As shown in Fig. 7, after the agent learns how to manage new abstract pains, total average pain stabilizes and moves toward equilibrium. (The dashed lines around each solid line correspond to a 95% confidence interval, determined by 2× the standard deviation of each average pain. These statistics represent results obtained after ten simulation runs.) As we can see, the exhaustive OML algorithm provides the best results, although, the difference among the linear, quadratic, and exhaustive algorithms is negligible in this case. However, when we run the simulation with a more complex environment consisting of eight primitive and 14 abstract needs (compared with five primitive and eight abstract needs in the baseline scenario), a greater performance benefit of the linear OML algorithm over the exhaustive OML can be observed in terms of computation time (Table VIII). Considering that the exhaustive algorithm is NP complete, the high effectiveness of the linear algorithm eliminates the need for more advanced decision-making process in most practical situations.

Table VIII shows how time requirements for the ML and OML algorithms depend on the number of pains used in the simulation. For the simulation results shown in Fig. 7, the exhaustive algorithm took on average 74.1 s compared with the 37.3 s needed for the linear OML implementation, and 32.6 s for the ML algorithm. In such a simple environment, the exhaustive OML algorithm did not take a large amount of time owing to only a few pains at a time being above threshold (3 on average), thereby keeping the computational depth of the exhaustive algorithm's factorial level complexity low.

However, in the more complex environments, more pains reside above threshold, increasing the computational time of the exhaustive algorithm as seen in last three rows in Table VIII.

While the ML and Linear OML runs have a linear dependency on the number of pains in the environment, the factorial

TABLE VIII
TIME NEEDED (s) VERSUS NUMBER AND TYPE OF PAINS

# AND TYPE OF PAINS	ML	LIN. OML	QD. OML	EX. OML
5 PRIM, 8 ABSTRACT	32.6	37.3	40.3	74.1
6 PRIM, 10 ABSTRACT	34.4	39.2	47.0	217.0
7 PRIM, 12 ABSTRACT	38.0	45.0	52.3	312.9
8 PRIM, 14 ABSTRACT	43.7	54.7	67.3	4633.0

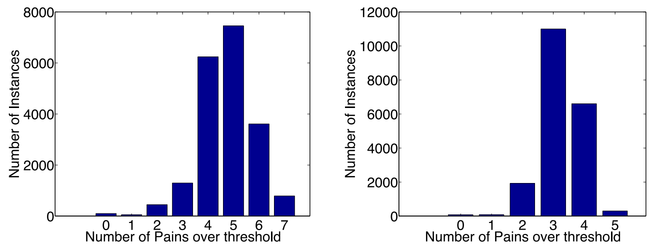

Fig. 8. Frequency of pains above threshold. Left: ML. Right: linear OML.

dependency of the exhaustive OML on the number of pains above threshold observed in Table VIII is definitely a problem for real-time applications.

It is clear that many more pains ended up above the threshold in the last three rows in Table VIII than in the baseline scenario (row one), increasing the time needed for exhaustive OML. Fig. 8 shows a pair of histograms showing the frequency for each number of pains above threshold (in the base line scenario with five primitive pains and eight possible abstract pains) for the standard ML and linear OML runs, respectively. From these charts, there is a clear indication that the OML algorithm does a better job managing the pains as it is able to keep more pains below threshold.

In the simulations, the agent does not automatically know how long the execution of each type of action will take before it tries it out, so it has to estimate. The agent assumes constant values to estimate the unfamiliar task's working time (shown as the estimated motor effort in Table IX). As it performs each task, the agent learns all the real working times. The real working times are represented such that they follow a rough Gaussian distribution with mean and standard deviation equal to 3 and 1, respectively. Table IX chronicles how the estimated effort values affect the performance of the algorithm by showing their respective average pain levels at the end of simulation runs.

From Table IX, we can see that the best results generally occur when the agent matches or underestimates the working time (except with the first instance of basic ML), as inferred by the fact that the lowest difference between linear OML and exhaustive OML occurs when the unfamiliar task working time is 2. This has the effect of biasing the agent toward more exploration.

So far, we have only looked at the clear advantages of implementing OML; however, are there any disadvantages? The most obvious one to consider is the increase in computational overhead associated with the OML algorithms. However, when observing the significant average pain reduction between ML and Linear OML and small computation time, this increase in

TABLE IX
TOTAL AVERAGE PAINS AT RUN COMPLETION
FOR VARYING ESTIMATED MOTOR EFFORTS

ESTIMATED MOTOR EFFORT	FINAL AVERAGE PAIN				DELTA LIN-EX.
	ML	LIN. OML	QUAD. OML	EX. OML	
1	6.8147	1.7688	4.4014	1.5250	0.2438
2	3.5543	1.8897	2.0354	1.8780	0.0117
3	3.5474	1.9224	2.0927	1.9103	0.0121
4	3.6028	1.9373	2.0935	1.9250	0.0123
5	3.6330	1.9039	2.0387	1.8860	0.0179
6	3.6231	1.8876	2.0199	1.8517	0.0359
10	3.6286	1.9112	2.0261	1.8819	0.0293
15	3.6536	1.9285	2.0699	1.9057	0.0228
20	3.6441	1.9152	2.0939	1.9125	0.0027

computational time is negligible. Another possible disadvantage is lack of any analysis of the past events or estimation of the future pains to better select opportunities to explore. However, this is something we are actively working on, moving toward dealing with more complex cognitive agents.

VI. CONCLUSION

In this paper, we have introduced the concept of an opportunistic intelligent agent improving our previous ML work. In earlier works [14]–[16], it was discussed why we believe that ML can surpass other approaches, in particular, standard RL. In this paper, we discussed how opportunistic behavior improves standard ML, and demonstrated results of OML. Additionally, we have discussed the importance of testing embodied agents in increasingly complex environments and provided two examples of our paper through Blender and NeoAxis implementations of ML and OML.

In our prior tests, we demonstrated that ML agents have been able to outperform similarly configured RL-based agents with no internally created goals. In this paper, it has been demonstrated that the opportunistic agent introduced in this paper can perform even better than the standard ML.

REFERENCES

[1] R. Pfeifer and J. C. Bongard, *How the Body Shapes the Way We Think: A New View of Intelligence*. Cambridge, MA, USA: MIT Press, 2007.
[2] R. S. Sutton, "Learning to predict by the method of temporal differences," in *Machine Learning*, vol. 3. New York, NY, USA: Springer-Verlag, 1988, pp. 9–44.
[3] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: MIT Press, 1998.
[4] A. G. Barto, S. Singh, and N. Chentanez, "Intrinsically motivated learning of hierarchical collections of skills," in *Proc. 3rd Int. Conf. Develop. Learn.*, San Diego, CA, USA, 2004, pp. 112–119.
[5] R. A. Brooks, "Intelligence without representation," *Artif. Intell.*, vol. 47, nos. 1–3, pp. 139–159, 1991.
[6] R. Pfeifer and C. Scheier, *Understanding Intelligence*. Cambridge, MA, USA: MIT Press, 1999.
[7] J. Schmidhuber, "Curious model-building control systems," in *Proc. IEEE Int. Joint Conf. Neural Netw.*, Singapore, Nov. 1991, pp. 1458–1463.
[8] W.-T. Fu and J. R. Anderson, "Solving the credit assignment problem: Explicit and implicit learning of action sequences with probabilistic outcomes," in *Psychological Res.*, vol. 72, no. 3, pp. 321–330, 2006.
[9] B. Bakker and J. Schmidhuber, "Hierarchical reinforcement learning based on subgoal discovery and subpolicy specialization," in *Proc. 8th Conf. Intell. Auto. Syst. (IAS)*, Amsterdam, The Netherlands, 2004, pp. 438–445.

[10] R. W. White, "Motivation reconsidered: The concept of competence," *Psychol. Rev.*, vol. 66, no. 5, pp. 297–333, 1959.

[11] P.-Y. Oudeyer, F. Kaplan, and V. Hafner, "Intrinsic motivation systems for autonomous mental development," *IEEE Trans. Evol. Comput.*, vol. 11, no. 2, pp. 265–286, Apr. 2007.

[12] K. E. Merrick, "A comparative study of value systems for self-motivated exploration and learning by robots," *IEEE Trans. Auto. Mental Develop.*, vol. 2, no. 2, pp. 119–131, Jun. 2010.

[13] L. Steels, "The autotelic principle," in *Embodied Artificial Intelligence* (Lecture Notes in Computer Science), vol. 3139. Berlin, Germany: Springer-Verlag, 2004, pp. 231–242.

[14] J. A. Starzyk, "Motivation in embodied intelligence," in *Frontiers in Robotics, Automation and Control*. Vienna, Austria: I-Tech Education & Publishing, Oct. 2008, pp. 83–110.

[15] J. A. Starzyk, "Motivated learning for computational intelligence," in *Computational Modeling and Simulation of Intellect: Current State and Future Perspectives*, B. Igelnik, Ed. Hershey, PA, USA: IGI Global, 2011, ch. 11, pp. 265–292.

[16] J. A. Starzyk, J. T. Graham, P. Raif, and A.-H. Tan, "Motivated learning for the development of autonomous systems," *Cognit. Syst. Res.*, vol. 14 no. 1, pp. 10–25, 2012.

[17] M. Schut and M. Wooldridge, "Principles of intention reconsideration," in *Proc. 5th Int. Conf. Auto. Agents*, New York, NY, USA, 2001, pp. 340–347.

[18] P. P. Saffiotti, S. Parsons, O. Pettersson, A. Saffiotti, and M. Wooldridge, "Robots with the best of intentions," in *Artificial Intelligence Today*. Berlin, Germany: Springer-Verlag, 1999, pp. 329–338.

[19] P. Anselme, "Opportunistic behavior in animals and robots," *J. Experim. Theoretical Artif. Intell.*, vol. 18, no. 1, pp. 1–15, 2006.

[20] T. Kruse and A. Kirsch, "Towards opportunistic action selection in human-robot cooperation," in *Proc. 33rd Annu. German Conf. Adv. Artif. Intell. (AI)*, Karlsruhe, Germany, Sep. 2010, pp. 374–381.

[21] R. H. Thomason, "Desires and defaults: A framework for planning with inferred goal," in *Proc. KR*, 2000, pp. 702–713.

[22] C. da Costa Pereira and A. G. B. Tettamanzi, "Goal generation with relevant and trusted beliefs," in *Proc. 7th Int. Conf. Auto. Agents Multiagent Syst. (AAMAS)*, 2008, pp. 397–404.

[23] J. Broersen, M. Dastani, J. Hulstijn, and L. van der Torre, "Goal generation in the BOID architecture," *Cognit. Sci. Quart.*, vol. 2, nos. 3–4, pp. 428–447, 2002.

[24] A. H. Maslow, *Motivation and Personality*. London, U.K.: Pearson, 1997.

[25] M. Hutter, "Towards a universal theory of artificial intelligence based on algorithmic probability and sequential decisions," in *Proc. 12th Eur. Conf. Mach. Learn. (ECML)*, 2001, pp. 226–238.

[26] K. E. Merrick, "Intrinsic motivation and introspection in reinforcement learning," *IEEE Trans. Auto. Mental Develop.*, vol. 4, no. 4, pp. 315–329, Dec. 2012.

[27] A. S. Rao and M. P. Georgeff, "BDI agents: From theory to practice," in *Proc. 1st Int. Conf. Multiagent Syst. (ICMAS)*, 1995, pp. 312–319.

[28] M. Dastani, F. Dignum, and J.-J. Meyer, "Autonomy and agent deliberation," in *Agents and Computational Autonomy: Potential, Risks, and Solutions*, vol. 2969, N. Nickels, M. Rovatsos and G. Weiss, Eds. Heidelberg, Germany: Springer-Verlag, 2004, pp. 114–127.

[29] F. Dignum and R. Conte, "Intentional agents and goal formation," in *Intelligent Agents IV: Agent Theories, Architectures and Languages*. Berlin, Germany: Springer-Verlag, 1998, pp. 231–243.

[30] M. Wooldridge, "Reasoning about rational agents," in *Intelligent Robots and Autonomous Agents*. Cambridge, MA, USA: MIT Press, 2000.

[31] J. Graham, J. A. Starzyk, and D. Jachyra, "Opportunistic motivated learning agents," in *Proc. 11th Int. Conf. ICAISC*, Apr./May 2012, pp. 442–449.

[32] J. A. Starzyk and D. K. Prasad, "A computational model of machine consciousness," *Int. J. Mach. Consciousness*, vol. 3, no. 2, pp. 255–281, 2011.

[33] N. Christofides, "Worst-case analysis of a new heuristic for the traveling salesman problem," GSIA, Carnegie Mellon Univ., Pittsburgh, PA, USA, Tech. Rep. 388, Feb. 1976.

[34] G. Cornuejols and G. L. Nemhauser, "Tight bounds for Christofides' traveling salesman heuristic," *Math. Program.*, vol. 14, no. 1, pp. 116–121, 1978.

[35] J. Bach, *Principles of Synthetic Intelligence PSI: An Architecture of Motivated Cognition* (Cognitive Models and Architectures). London, U.K.: Oxford Univ. Press, Apr. 2009.

[36] K. E. Merrick and M. L. Maher, *Motivated Reinforcement Learning: Curious Characters for Multiuser Games*. New York, NY, USA: Springer-Verlag, Jun. 2009.

[37] Linden Lab. *Second Life*. [Online]. Available: http://www.secondlife.com, accessed Oct. 2012.

[38] G. Metta *et al. The iCub Humanoid Robot: An Open Platform for Research in Embodied Cognition*. [Online]. Available: http://www.icub.org, accessed Jan. 22, 2013.

[39] J. Graham and J. A. Starzyk, "Transitioning from motivated to cognitive agent model," in *Proc. IEEE Symp. Comput. Intell. Human-Like Intell.*, Singapore, Apr. 2013, pp. 9–16.

[40] *NeoAxis—All-Purpose, Modern 3D Graphics Engine for 3D Simulations, Visualizations and Games*. [Online]. Available: http://www.neoaxis.com/, accessed Feb. 10, 2013.

James Graham (M'09) received the B.S.E.E. degree in electrical engineering from the University of Michigan, Ann Arbor, MI, USA, in 2001, and the M.S.E.E. degree from Ohio University, Athens, OH, USA, in 2007, where he is currently pursuing the Ph.D. degree with the School of Electrical Engineering and Computer Science.

His current focus is the development of a cognitive architecture based on motivated learning. His current research interests include cognitive intelligence, machine learning, pattern recognition, and tracking.

Janusz A. Starzyk (SM'83) received the M.S. degree in applied mathematics and the Ph.D. degree in electrical engineering from the Warsaw University of Technology, Warsaw, Poland, and the Habilitation degree in electrical engineering from the Silesian University of Technology, Gliwice, Poland.

He was an Assistant Professor with the Institute of Electronics Fundamentals, Warsaw University of Technology. He spent two years as a Post-Doctoral Fellow and a Research Engineer with McMaster University, Hamilton, ON, Canada. Since 1991, he has been a Professor of Electrical Engineering and Computer Science with Ohio University, Athens, OH, USA, and the Director of Embodied Intelligence Laboratories. Since 2007, he has been the Head of the Information Systems Applications with the University of Information Technology and Management, Rzeszow, Poland. He has cooperated with the National Institute of Standards and Technology, Gaithersburg, MD, USA, where he has been involved in the area of testing and mixed-signal fault diagnosis for eight years. His current research interests include embodied machine intelligence, motivated goal-driven learning, self-organizing associative spatiotemporal memories, active learning of sensorymotor interactions, machine consciousness, and applications of machine learning to autonomous robots and avatars.

Daniel Jachyra (M'10) received the M.S. degree in informatics technologies from the University of Information Technology and Management, Rzeszow, Poland, in 2007, where he is currently pursuing the Ph.D. degree.

He is currently a Computational Cognitive Neuroscience Researcher and a EUCogIII Member of a European network of nearly 900 researchers in artificial cognitive systems and related areas who want to connect to other researchers. His current research interests include mechanisms of motivated behaviors, goal creation, machine learning in autonomous agents, and long-term memory structures.