Georgeon, O. L., Morgan, J. H., & Ritter, F. E. (2010). An algorithm for self-motivated hierarchical sequence learning.
In D. D. Salvucci & G. Gunzelmann (Eds.), Proceedings of ICCM - 2010- Tenth International Conference on Cognitive Modeling (pp. 73-78).

An Algorithm for Self-Motivated Hierarchical Sequence Learning

Olivier L. Georgeon (olg1@psu.edu)
Jonathan H. Morgan (jhm5001@psu.edu)
Frank E. Ritter (frank.ritter@psu.edu)

The College of Information Sciences and Technology
The Pennsylvania State University, University Park, PA 16802

Abstract

This work demonstrates a mechanism that autonomously organizes an agent's sequential behavior. The behavior organization is driven by pre-defined values associated with primitive behavioral patterns. The agent learns increasingly elaborated behaviors through its interactions with its environment. These learned behaviors are gradually organized in a hierarchy that reflects how the agent exploits the hierarchical regularities afforded by the environment. To an observer, the agent thus appears to exhibit basic self-motivated, sensible, and learning behavior to fulfill its inborn predilections. As such, this work illustrates Piaget's theories of early-stage developmental learning.

Keywords: Developmental learning; cognitive architectures; situated cognition; computer simulation.

Introduction

We report the implementation of an agent that autonomously engages in a process of hierarchical organization of behavioral schemes as it interacts with its environment. This mechanism moves towards taking on developmental constraints as Newell (1990, p. 459+) called for, and generates high-level and long-term individual differences in representation and behavior that arise from lower level behavior.

This implementation also refers to an "emergentist" and a constructivist hypothesis of cognition. According to these hypotheses, an observer can attribute cognitive phenomena (such as *knowing*, *feeling,* or *having motivations*) to the agent while observing its activity, provided that the agent's behavior can appropriately organize itself. These hypotheses have often been related to Heidegger's philosophy of mind, e.g., cited by Sun (2004). Additionally, these hypotheses correspond to work featuring constructivist epistemologies (Le Moigne, 1995; Piaget, 1937), situated cognition (Suchman, 1987), and embodied cognition (Wilson, 2002).

We describe the agent as self-motivated because it does not seek to solve a problem pre-defined by the modeler, nor does it learns from a reward that is given when reaching a pre-defined goal. Rather, the agent learns to efficiently enact its inborn predilections by exploiting regularities it finds through its activity. As such, the implementation constitutes a model of agents exhibiting intrinsic motivation, pragmatic and evolutionist learning, as well as sensible behavior.

To situate the technical approach in the field of artificial intelligence, we can refer to Newell and Simon's (1975) physical symbol hypothesis. We subscribe to the hypothesis's weak sense. We are using computation to generate intelligent behavior. We, however, do not subscribe to the hypothesis's strong sense, in that we are not implementing symbolic computation based on symbols to which we would pre-attribute a denotation. Instead, we will discuss how knowledge appears to emerge (to an external observer) from the agent's activity, and how the agent seems to make sense of the knowledge because it is grounded in the agent's activity (Harnad, 1990).

Although we did not follow a symbolic computational modeling approach, we have, nevertheless, implemented this model in a cognitive architecture, namely Soar 9. We chose Soar because it has proven efficient for implementing mechanisms for behavior organization. In particular, we found Soar 9's mechanisms for graph querying and operator selection based on valued preferences very helpful.

Knowledge representation

The agent's behavioral patterns are represented using two kinds of objects: *schemas* and *acts*. We use the term schema in its Piagetian (1937) sense, meaning a behavioral pattern or sensorimotor pattern. An *act* is a notion specific to our work that refers to a schema's enaction. By schema's enaction, we mean the association of a schema with the feedback the agent receives when enacting the schema. Concretely, an act associates a schema with a binary feedback status: *succeed (S)* or *fail (F)*. Hence, each schema is associated with at most two acts: its *failing* act and its *succeeding* act. Schemas and acts are organized in a hierarchy as shown in Figure 1.

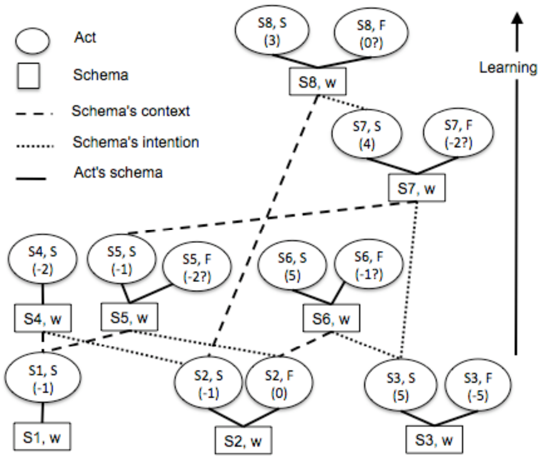

Figure 1: Example schema and act hierarchy.

At its lowest level, Figure 1 shows primitive schemas S1, S2, and S3. Primitive schemas define the agent's primitive possibilities of behavior within a given environment. For example, as further detailed in the experiment section, S1 may correspond to *turn right*, S2 *touch ahead*, and S3 *attempt to move forward*. Primitive acts are represented above primitive schemas. For example, act [S3, S, 5] corresponds to *succeeding in moving forward*, while [S3, F, -5] corresponds to *bumping into a wall*. Each act has a value associated with it, in this case: 5 and -5 (in parentheses in the figure). These values inform the selection of the next schema to enact, as explained later. For now, we can understand these values as the agent's *satisfaction* for performing the act.

Primitive satisfaction values are chosen and hard-coded by the modeler according to the behavior she intends to generate. In our example, act [S3, S, 5] means that the agent *enjoys* moving forward, while act [S3, F, -5] means that the agent *dislikes* bumping into walls. Similarly, act [S2, S, -1] means that the agent *touches a wall* in front of him, which he slightly dislikes; while [S2, F, 0] means that the agent *touches an empty square*, which leaves him indifferent. Therefore, primitive satisfaction values are also a way for the modeler to define the agent's intrinsic motivations.

Higher-level schemas are learned through experience, by combining lower level schemas. Schema learning consists of adding the new-learned schema to the agent's memory as a node and two arcs pointing to the schema's sub-acts. For example, schema S5 is learned when the agent has turned to the right and then touched an empty square. Schemas have a *context act* (dashed line in the figures throughout this paper), an *intention act* (doted line), and a weight (w). So, S5 means that, when the agent has successfully turned right, the agent can expect to touch an empty square. Similarly, S4 is learned when the agent has successfully turned right and touched a wall. S4 thus generates the opposite expectation from S5. A schema's weight corresponds to the number of times the schema has been enacted. Over the course of the agent's interactions, the relative schema weights thus balance the agent's expectations in specific contexts.

When a higher-level schema is learned, its succeeding act is also learned with a satisfaction value set equal to the sum of the satisfaction values of its sub-acts, e.g., [S4, S, -2] (-1-1) and [S5, S, -1] (-1+0). When a higher-level schema gains enough weight, it can be selected for enaction. Enacting a higher-level schema consists of sequentially enacting its sub-acts. For example, enacting S5 consists of enacting S1 with a succeeding status, then enacting S2 with a failing status. Hence, the satisfaction for enacting a high-level act is equal to the satisfaction for individually enacting its sub-acts.

When a high-level schema fails during enaction, it is interrupted. This happens if a status returned by the environment does not match the expected status of a sub-act. In this case, the failing act of the schema is learned or reinforced, as well as the *actually enacted act*. The satisfaction value of the failing act is set equal to the satisfaction value of the actually enacted act. For example, if schema S6 fails because S2 succeeds, then [S6, F, -1] is learned. Because high-level schemas can potentially fail at any step of their sequence, their failing act's satisfaction values are averaged over their different failures.

When a high-level schema is enacted, it generates the learning of higher schemas. For example, when S5 is successfully enacted and followed by succeeding S3, then S7 is learned. In this example, S7 consists of turning right, touching an empty square, and then successfully moving forward. [S7, S]'s satisfaction is set equal to 4 (-1 + 5). Similarly, S8 (learned after S7) consists of touching a wall, turning right, touching an empty square, and moving forward.

Algorithm

The algorithm follows two overlapping cyclical loops. The *control loop* consists of: 1: selecting a schema for enaction, 2: trying to enact the selected schema, 3: learning what can be learned from this trial, 4: computing the resulting situation, and finally looping to step 1. We call this loop the control loop because it is at this level that the agent *decides* what schema to try to enact.

Step 2: (trying to enact a schema) constitutes a nested loop that goes through the selected schema's hierarchical structure and tries to enact each of its primitive acts sequentially. We call this loop the *automatic loop* because this loop enacts sub-schemas below the agent's decision process. Figure 2 illustrates this procedure.

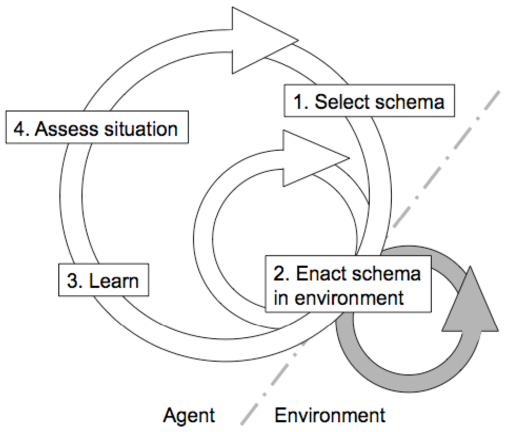

Figure 2: Algorithm procedure.

In Figure 2, the large white circle represents the control loop while the small white circle represents the automatic loop. The gray circle represents the environment's loop. Each revolution of the automatic loop corresponds to a revolution of the environment's loop that returns the status of the enacted primitive schema. From the viewpoint of the control loop, the schema's enaction constitutes only one step, whatever the schema level is in the hierarchy. Therefore, at the control loop level, any schema is handled similarly as a primitive schema, which makes possible the recursive learning of higher-level schemas.

The four steps of the control loop are:

Step 1: All schemas whose *context act* matches the previously assessed situation propose their *intention act*. The weight of this proposition is computed as the proposing schema's weight multiplied by the intention act's satisfaction. The schema with the highest proposition is selected (if several schemas are equal, one is randomly picked among them). In essence, this mechanism selects the schema that will result in the expected act having the highest satisfaction, balanced by the probability to obtain this expected act. This probability is based on what the agent has learned thus far concerning the current context. Due to this mechanism, the agent appears (to an observer) as though he was seeking to enact the act associated with the highest *believed* expected satisfaction and avoiding the acts with the lowest ones. Figure 3 illustrates this mechanism.

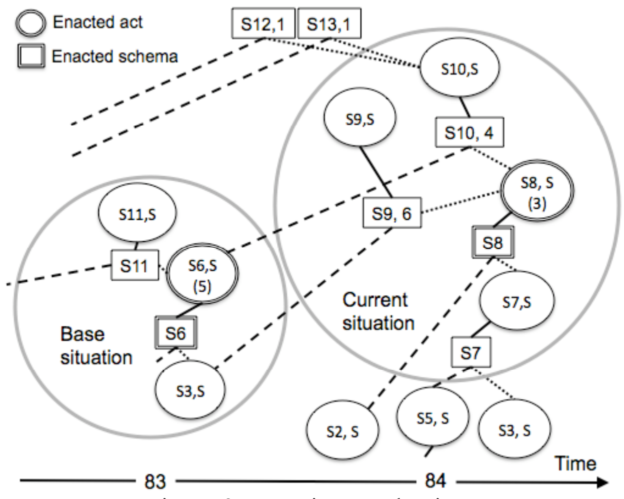

Figure 3: Enaction mechanism.

Figure 3 details the 84th iteration of the control loop in the experiment reported in Figure 5. On the 83rd iteration, schema S6 was successfully enacted (touch empty square, move forward), which resulted in a base situation of [S6, S], [S3, S], and [S11, S] (and other acts on top of [S6, S] not reported in the figure). In this context, S9 and S10 were activated and proposed to enact S8 with a proposition weight of 6x3+4x3 (sum of the proposing schema's weight multiplied by [S8, S]'s satisfaction) (the agent never experienced S8 failing). This proposition happened to be the highest of all the propositions, which resulted in S8 being selected for enaction.

Step 2: The algorithm next enacts all the selected schema's sub-acts. If all the sub-acts meet their expectations, the control loop proceeds to step 3. If the enaction of one of the sub-acts is incorrect, then the automatic loop is interrupted; the schema's enaction status is set to fail; and control is returned to the control loop. In Figure 3's example, the enaction of schema S8 consists of the enaction of acts [S2, S], [S5, S] (made of [S1, S] and [S2, F], as shown in Figure 1), and [S3, S] in a sequence. In this case, S8 was successfully enacted, resulting in the enacted act [S8, S].

Step 3: New schemas are learned or reinforced by combining the base situation and the current situation. In Figure 3's example, S9's weight is incremented from 6 to 7, and S10's weight is incremented form 4 to 5. In addition, new schemas are learned based on the penultimate situation and on [S10, S] (e.g., S12 and S13 are created with a weight of 1, as well as other schemas not represented in the figure).

Step 4: The *base situation* becomes the *penultimate situation* and the *current situation* becomes the base situation for the next cycle. A *situation* is made of the acts that surround the enacted act (i.e., the enacted act, the acts directly below it, and the acts directly above it). In Figure 3's example, the *situation* is made of [S8, S], [S7, S], [S9, S], and [S10, S]. The *situation* can be understood as the agent's *situation awareness*, that is, a representation of the agent's situation in terms of affordances (Gibson, 1979) capable of activating behavior. Limiting the situation to the acts directly surrounding the enacted act prevents the agent from being overwhelmed by a combinatorial explosion as the agent creates new schemas. In essence, the agent focuses on the current level of *abstraction* for representing his situation, for making his choices, and for finding and learning higher-level regularities. When a high-level schema fails during enaction, the agent constructs the *actually enacted* schema and falls back to a lower abstraction level.

Experiment

To test the algorithm, we developed an environment that afforded the agent hierarchical sequential regularities to learn and organize. Although the interaction's structure—resulting from the coupling of the environment with the agent's primitive schemas—is fundamentally sequential, the environment appears to external observers as a two-dimensional grid represented in Figure 4, implemented from Cohen's (2005) Vacuum environment.

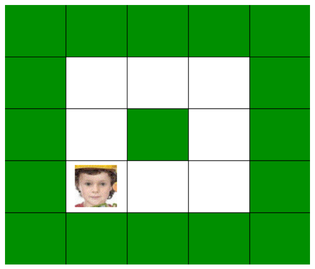
Figure 4: Experimental environment.

In Figure 4, white squares represent empty squares where the agent can go, and filled squares represent walls. The agent's primitive schemas and acts are defined as described above (S1=turn 90° right (-1/NA), S2=touch the square ahead (-1/0), S3=attempt to move one square forward (5/-5)). Additionally, we have primitive schema S0 consisting of turning to the left (-1/NA) (turning schemas S0 and S1 always succeed in this environment). These settings offer a first notable regularity, namely that the agent can increase his average satisfaction by touching ahead before trying to move forward, and not moving forward if he touches a wall.

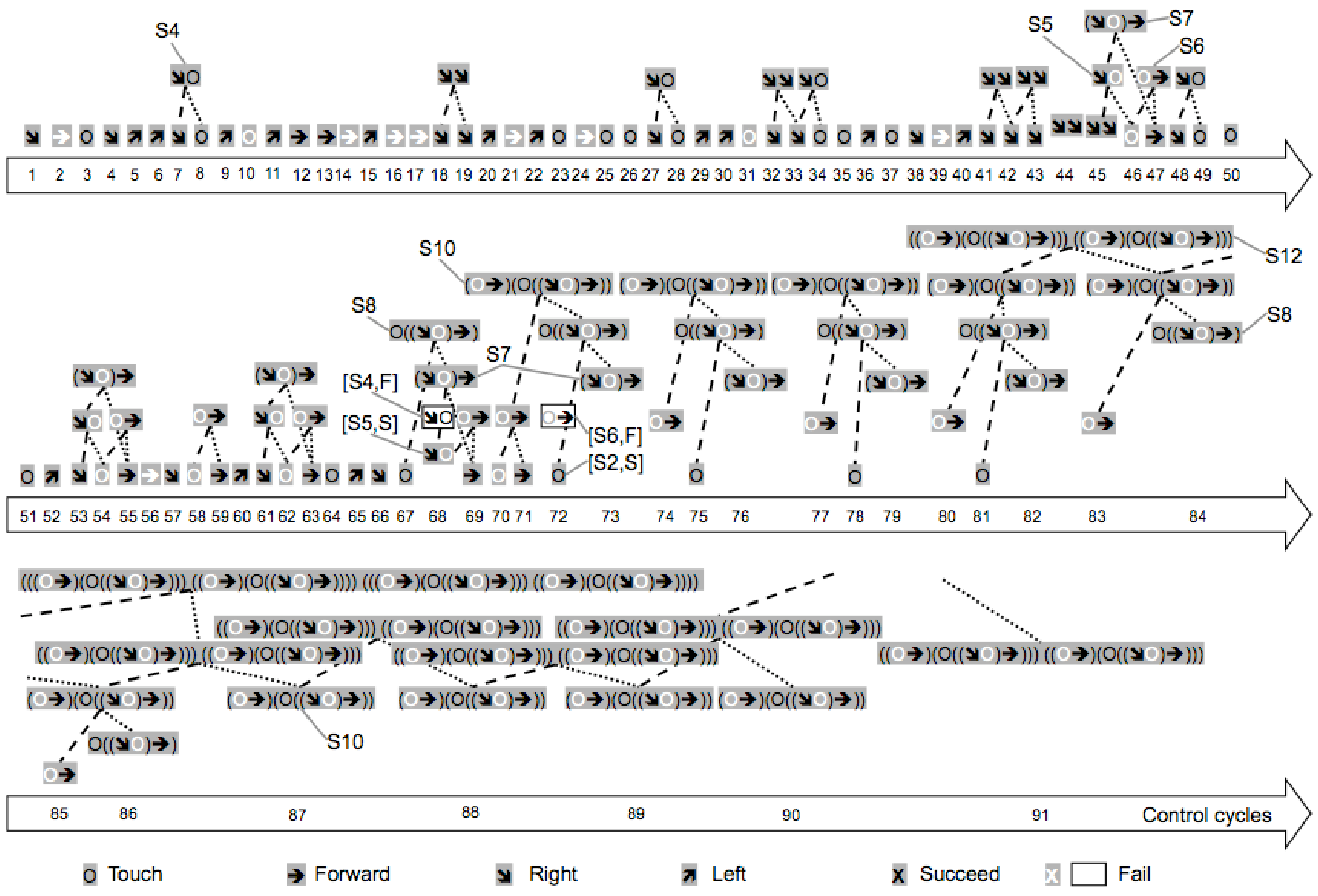

Figure 5: An example run among the 18 reported in row 6 of Table 1.

Next, the agent can increase his satisfaction by repeating the sequence consisting of moving forward twice and turning once. Higher-level regularities consist of repeating this later sequence. The effects of this learning mechanism are shown in detail in Figure 5 that reports an example run. Videos of other runs can be seen online[1].

In Figure 5, an attempt to move forward is represented as an arrow to the right, a turn-left as an upward arrow, a turn-right as a downward arrow, a touch as a O. Succeeding primitive schemas use a black font, while failing primitive schemas use a white font, i.e., white rightward arrows mean that the agent bumped into a wall, and white Os mean that the agent touched an empty square in front of him. Enacted schemas are represented at the lowest level in each line while learned schemas are represented on top of the enacted schemas. Failing higher-level schemas are represented as white boxes with black outlines (steps 68 and 72). The numbers from 1 to 91 indicate the control-loop iterations (steps).

At the beginning, the agent acts randomly because he has not yet learned appropriate schemas that could propose their associated intention sub-schema. However, every cycle, the agent constructs or reinforces several schemas. For clarity, Figure 5 only reports the construction and the reinforcement of the schemas that matter for the purpose of explanation, and references these schemas when they are mentioned in the text. Schema S4 is constructed on step 8. S4 is then reinforced on step 28, 34, and 49. The agent attempts to enact S4 for the first time on step 68 but fails and enacts S5 instead.

Notably, a schema *turn right-turn right* (not named in this paper) is constructed on step 19. This schema is reinforced on steps 33, 42, and 43. It is then enacted twice on steps 44 and 45. It is, however, not used any further because other options prove more satisfying (its satisfaction value is -2).

On step 46, the agent constructs the schema S5 (using act [S1, S] that is the schema *turn right-turn right*'s intention act). Then, on step 47, the agent finds the schema S6 (touch empty, move forward), and also constructs the schema S7 on top of S5. After step 47, the schema S6 always prompts the agent to try to move forward after touching an empty square; therefore, from then on, S6 is quickly reinforced in steps 55, 59, 63, and 71. The agent tries to enact S6 for the first time on step 72, but unsuccessfully, which results in falling back to [S2, S]. This experience instructed the agent that schema S6's failing act has a satisfaction of -1, which is still better than trying to move forward without touching first and bumping into a wall (satisfaction -5). Therefore, from then on, the agent learned to touch before moving

[1] http://e-ernest.blogspot.com/2009/07/ernest-64.html

forward. S6 is then successfully enacted on steps 74, 77, 80, 83, and 85.

As said previously, on step 68, the agent intended to enact S4 but actually enacted S5. Because S7 is directly above enacted schema S5, S7 is included in the agent's situation awareness, which results in the learning of the fourth-order schema S8 on step 69. Then, on step 73, the enaction of schema S7 generated the learning of schema S10. As detailed in Figure 3, S8 is enacted for the first time on step 84, which generated the learning of S12. S10 starts to be enacted on step 87.

After step 87, the agent keeps on performing the sequence *touch empty, move forward, touch wall, turn right, touch empty, move forward.* This regularity introduces repeated circuits that lead to higher-level repetitions of this sequence. With this sequence, the agent obtains a satisfaction of 8 within 6 primary steps, i.e., 1.33 per primary step.

In this example, the agent did not learn the optimum sequence in the environment. In fact, the agent has no way to know whether the stabilized sequence is optimum or not. The agent only repeats a sequence when other actions appear less likely to bring satisfaction, based on what he has learned before. In most instances, the agent first learns to touch before moving, after which he begins to build other regularities based on this initial pattern.

The experiment was run 100 times, stopping each run when the agent has reached a stable sequence, and clearing the agent's memory between each run. The results are summarized in Table 1.

Table 1: Summary of hundred runs.

Row	Runs	Satisfaction/step	Cycles
1	22	3.00	50
2	22	2.25	79
3	4	1.80	75
4	4	2.00	69
5	16	1.60	62
6	18	1.33	84
7	1	1.40	76
8	1	1.17	109
9	1	1.00	108
10	2	0.75	116
11	3	1.00	61
12	1	0.80	95
13	3	1.00	71
14	2	0.40	96
	100	1.92	72

In Table 1, the runs are aggregated by average satisfaction per step obtained when the agent has reached a stable sequence. The column *Cycles* reports the average number of control loop cycles before reaching this sequence. Rows 1 through 6 report 86 runs where the agent learned to go around his environment and got a satisfaction per step greater than or equal to 1.33. Rows 7 to 14 report 14 runs where the agent has stabilized on a sequence that results in staying on one edge of the environment, and reached a satisfaction per step that ranged between 0.40 and 1.40.

The summary row shows that the average reached satisfaction per step was of 1.92. It was reached in an average of 72 cycles. In comparison, other experiments yielded an average satisfaction values per step of -0.93 without any learning and -0.38 with only the first-level schema learning. This data demonstrates that, in all the runs, the hierarchical learning mechanism has substantially increased the agent's satisfaction, compared to no or non-hierarchical learning.

Related works

To our knowledge, this work constitutes the first implementation of an intrinsically motivated agent who recursively learns to exploit hierarchical sequential regularities to fulfill drives. The closest related work is probably Drescher's (1991) attempt to implement Piagetian constructivist learning through what he called the *constructivist schema mechanism*. Like our implementation, Drescher's work constructed hierarchical schemas that associated context, actions, and expectations. In Drescher's implementation, however, schemas were neither associated with satisfaction values nor did the agents exhibit self-driven behavior. The agent's exploration was rather random and resulted in a combinatorial explosion as the agent encountered increasingly complex environments.

Chaput (2004) proposed the Constructivist Learning Architecture (CLA) to address Drescher's scalability issues. The CLA implemented a scheme *harvesting* mechanism at each hierarchical level. This harvesting, however, depended on goals defined by the modeler. Chaput's solution, therefore, relies upon a problem-solving approach that in fact differs from our self-driven mechanism of interest.

In developmental robotics (Weng et al., 2001), the literature often refers to Brooks's (1991) pioneering work. For example, Blank, Kumar, Meeden, and Marshall (2005) describe the principles for a self-motivated/self-organizing robot. They use the robot's anticipation reliability as a motivational regulator for the robot. As opposed to our work, these implementations do not make explicit the robot's driving satisfaction values. They also rely on a limited number of predefined hard-coded hierarchical layers, which restricts the agent's learning possibilities.

As for the testbed environment and self-driven learning paradigm, our approach appears to be rather unique. We must note that our learning paradigm substantially differs from maze solving experiments (e.g., Sun & Sessions, 2000) or from hierarchical sequence learning as depicted in the classical taxi cab experiment (Dietterich, 2000). In these experiments, the learning occurs over multiple runs (often thousands), and comes from a reward value that is given when the goal is reached and then backward propagated during subsequent runs. On the contrary, in our paradigm, there is no final goal that would provide a reward; the learning occurs through each run; and all the agent's memory is reinitialized between each run (including all forms of reinforcement).

Discussion and conclusion

Besides the quantitative results of the agent's measured satisfaction and that it learns at a nice pace (neither one shot nor thousands shots learning), this work offers qualitative results in the form of the agent's exhibited behavior. When observing the agent, an observer can infer that the agent seems to enjoy certain behaviors (such as moving forward) and dislike others (such as bumping into walls). Moreover, the agent appears to learn to endure unpleasant behaviors (such as turning or touching) to have more opportunities to move forward. The agent thus appears to be self-motivated and appears to learn knowledge about his environment that he uses to satisfy his predilections. More elaborated behaviors can be watched in videos online[2].

In addition, the agent appears to learn to use certain schemas as perceptions (e.g., schema S2 to *sense* the square forward), and to determine subsequent actions based upon these schema's outcomes. Therefore, the agent seems to simultaneously learn to perceive his environment and to make sense of his perception. This result is original in that the agent's perception was not pre-defined by the modeler in the form of a perceptual buffer, as it is in many cognitive models. In our case, perception emerges from the agent's behavior, which grounds the meanings of the agent's perceptions in his activity.

Moreover, the agent constructs an internal data structure made of elaborated behavioral patterns, and uses this data structure to deal with his environment. The behavioral patterns used in this data structure are only those confirmed through experience, which helps the agent deal with the environment's complexity. These data structures can be seen as the agent's *situation awareness* that is constructed through his interactions, and that activates subsequent behavioral patterns based on expected enjoyment. At each point in time, the current agent's knowledge frames how the agent *sees* the world, which makes possible the recursive learning of higher-level regularities and which accounts for the agent's individualization through his development.

Preliminary experiments in more complex environments show that this algorithm faces two notable limitations. One limitation is that the algorithm may represent the same primitive sequence by different schemas that have different hierarchical structures. These different schemas are useful to find appropriate hierarchical regularities but they impede the agent's performance in more complex environments. Future studies should find a way to merge these schemas. The second limitation is that the algorithm is not good at finding spatial regularities. For example, if we replace the central wall square with an empty square, the agent becomes less likely to find the most satisfying regularity, that of making a continuous circuit around his environment.

We, nevertheless, believe that these limitations are not insurmountable, and we plan to gradually increase the complexity of the agent and of the environment in future studies. We will add new drives to the agent, for example homeostatic drives (similar to hunger) or boredom-avoidance based on top-level regularity detection. We will also add other primitive schemas, especially schemas associated with distal perception. These schemas should, we believe, help the agent deal with open spatial environments.

[2] http://e-ernest.blogspot.com/2009/10/enrest-72.html

Acknowledgments

Support was provided by ONR (N00014-06-1-0164 and N00014-08-1-0481) and DTRA (HDTRA 1-09-1-0054).

References

Blank, D. S., Kumar, D., Meeden, L., & Marshall, J. (2005). Bringing up robot: Fundamental mechanisms for creating a self-motivated, self-organizing architecture. *Cybernetics and Systems, 32*(2).

Brooks, R. (1991). Intelligence without representation. *Artificial Intelligence Journal, 47*, 139–159.

Chaput, H. H. (2004). *The Constructivist Learning Architecture: A model of cognitive development for robust autonomous robots.* Unpublished doctoral dissertation, The University of Texas, Austin.

Dietterich, T. G. (2000). *An Overview of MAXQ Hierarchical Reinforcement Learning*. Paper presented at the SARA02 4th International Symposium on Abstraction, Reformulation, and Approximation.

Drescher, G. L. (1991). *Made-up minds, a constructivist approach to artificial intelligence*. Cambridge, MA: MIT Press.

Gibson, J. J. (1979). *The ecological approach to visual perception*. Boston: Houghton-Mifflin.

Harnad, S. (1990). The symbol grounding problem. *Physica, D*(42), 335-346.

Le Moigne, J.-L. (1995). *Les épistémologies constructivistes*. Paris: Presse Universitaire de France.

Newell, A. (1990). *Unified Theories of Cognition*. Cambridge, MA: Harvard University Press.

Newell, A., & Simon, H. (1975). Computer science as empirical inquiry: symbols and search. *Communications of the ACM, 19*(3), 113-126.

Piaget, J. (1937). *The construction of reality in the child*. New York: Basic Books.

Suchman, L. A. (1987). *Plans and situated actions*. Cambridge: Cambridge University Press.

Sun, R. (2004). Desiderata for cognitive architectures. *Philosophical Psychology, 17*(3), 341-373.

Sun, R., & Sessions, C. (2000). Automatic segmentation of sequences through hierarchical reinforcement learning. In R. Sun & C. L. Giles (Eds.), *Sequence Learning* (pp. 241–263). Berlin Heidelberg: Springer-Verlag.

Weng, J., McClelland, J., Pentland, A., Sporns, O., Stockman, I., Sur, M., et al. (2001). Artificial intelligence - Autonomous mental development by robots and animals. *Science, 291*(5504), 599-600.

Wilson, M. (2002). Six views of embodied cognition. *Psychonomic Bulletin & Review, 9*(4), 625-636.