Learning and Action Correction in Routine Tasks: New Changes to the Contention Scheduling Model

An Honors Thesis for the Department of Psychology

Matthew Taylor

Tufts University, 2012

Author Note

Matthew Taylor, Department of Psychology, Tufts University.

The author would like to thank Dr. Phil Holcomb for his help as a member of the author's thesis committee. The author would also like to thank Dr. Matthias Scheutz for all of his help in the formulation and oversight of this study.

Correspondence concerning this article should be addressed to Matthew Taylor, 744 Clarendon Road, Penn Valley, PA 19072. E-mail: mewtaylor@gmail.com.

Abstract

Sequential, routine tasks are part of our every day life – putting on clothes, driving to work, making coffee, making dinner, etc. They are the definition of ordinary and trivial goals. However, the amount of computation required by the brain to complete these actions is quite large, and consequently, the ability to simulate these actions on a computer has been limited up to now. One model, Contention Scheduling, is considered a successful computational model of routine task implementation, demonstrating ordinary action selection, as well as errors in implementation we observe in humans. However, the model has its own deficiencies with regards to learning and action correction. The purpose of this paper is to explore the use of Hebbian learning and bottom-up connections in modifying Contention Scheduling to resolve these issues. In order to do this, we implemented our own version of the original model, as well as versions including learning and action correction, and compared the results. Our results indicate that Hebbian learning complements the model well to simulate the cognition of procedural learning, and that bottom-up connections mitigate a large portion of the observed errors in the model.

Learning and Action Correction in Routine Tasks: New Changes to the Contention Scheduling Model

Everyday, we perform countless amounts of routine tasks as we move through our lives. We make coffee, take the train to work, cook dinner, work out at the gym… the list goes on and on. Not only do we perform these routine tasks, but we also integrate all of these actions together into sequences. Yet our ability to seamlessly go about these activities is not extraordinary to us – in fact, it is the definition of ordinary.

However, it takes our brains a substantial amount of computation to perform the actions necessary to complete these goals. In order to do something as simple as brush your teeth, you have to coordinate fine motor movements involved in brushing, while moving your hand to different parts of your mouth, all the while keeping track of where you have brushed, and where you have not. The computational power required to complete these actions is so large that we have only recently been able to replicate parts of these actions in non-biological life forms (i.e. robots, computer simulations, etc.).

On the other hand, when looking at the biological bases of these actions, the research looking into sequential tasks has, up until now, made substantial strides. Discoveries include the brain bases for sequential learning and implementation of routine tasks. However, our knowledge of how these ideas relate to the cognition of routine tasks is limited. Many theories have been proposed, but it has been difficult to prove or disprove them without computer models. Computer models enable the capacity to manipulate variables that are otherwise not possible to alter in humans without manipulating the brain itself.

Norman and Shallice (1986) were among the first to propose a computational model for how we store and retrieve sequential information in the brain, introducing a model called Contention Scheduling. However, their work was theoretical. During the 1990's, researchers began to explore using computers to simulate human cognition. The scope of these studies, however, was quite limited by the computational power available to researchers at the time. In 2000, Cooper and Shallice finally implemented the theoretical work of Norman and Shallice computationally. They used the Contention Scheduling model to simulate action selection both in ordinary people and in those suffering from Action Disorganization Syndrome, yielding results that mirrored the behavior of humans in both conditions.

Comment [FI1]: And what was their result? Did the model hold true?

Within the young field of Computational Psychology, Contention Scheduling is considered quite successful. In many ways, its hierarchical structure accurately reflects what researchers in neuroscience have discovered about the way the brain stores routine tasks. Further, it models errors in action selection that we see in our lives every day – forgetting to put the car into drive before pressing down on the gas, or getting distracted and continuously pouring sugar into the coffee cup accidentally. Intentionally modeling errors in computer simulations is challenging, which is part of the reason this model is considered so successful.

While Contention Scheduling has been viewed positively within the field of Computational Psychology, it still has its limitations. As outlined by Andronache and Scheutz (2002), it is a learned model – it includes no method for learning a new sequential task, but rather simulates tasks that have already been learned. On top of that, it does not have an action correction component. When an error occurs, the model does

not continue to work to completion, but rather remains in a state of incompletion. Both of these limitations, learning and action correction, are essential aspects of cognition. Sequential tasks are not stored in our brains from the time we are born – they are learned throughout life. If we could not correct ourselves when we made a mistake, that would put serious limitations on what we could accomplish day-to-day.

The purpose of this paper is to explore solutions to the limitations of Contention Scheduling outlined above. There are four components to the findings detailed in this paper. First, we recreated the original model of Contention Scheduling. Second, we altered the original model to incorporate a learning component. Third, we altered the model to include action correction components. Finally, we altered the model to include both learning and action correction.

In order to explore the effects of these components, we measured both the behavioral patters of the models as well as their time-scales for completion. For behavioral patterns, our hypothesis was that learning would have no effect on the model, but that action correction components would lead to fewer registered errors. For time-scale, our hypothesis was that action correction components would have no effect, whereas a learning model would require more time to work to completion. We hypothesized that there would be no interaction effects between learning and action correction.

We will start by outlining the original Contention Scheduling model from Cooper and Shallice (2000). Following that, we will describe the changes that we made to Contention Scheduling in order to explore action correction and learning. We will then

look at the findings of our implementation of the model. Finally, we discuss the impact of those results.

Background Information

Computational Models of Procedural Tasks to Date

We begin by examining how Computational Psychology has modeled procedural learning so far. Up until now, the number of computational models looking at procedural memory and procedural learning has been small. Rumelhart and Norman (1982) were among the first to look at a computational representation of sequential actions, and Norman and Shallice (1986) were the first ones to propose the use of schemas (described below) for action selection. However, as mentioned above, it was not until Cooper and Shallice implemented Contention Scheduling in 2000 that there was a complete model of routine action selection.

Contention Scheduling uses the Interactive Activation Competition (IAC) paradigm of neural network modeling to explore action selection in routine tasks. IAC models simulate neural activity with abstract "neurons", which can either activate or inhibit other neurons to which they are connected (this is what gives rise to the idea of "competition" within networks). However, the abstract neurons in this network do not correspond to neurons in the brain in a one-to-one ratio. Instead, each neuron represents a concept or action that is stored in the brain, and the network explores the interaction among these concepts. Contention Scheduling is considered one of the most successful IAC models to date.

Yet while it is considered a success, there are those that disagree with the hierarchical paradigm of the model. Botvinik and Plaut (2003) responded to Cooper and Shallice (2000) with a few pieces of criticism, mostly regarding Contention Scheduling's sequencing mechanism and its hierarchical structure. They instead implemented a recurrent connectionist approach to routine tasks. Their model, after significant training, was able to reproduce the behavior of Contention Scheduling using hidden layers as opposed to hierarchical structures. This model has been highlighted by some researchers as being more representative of neuropsychological phenomena than Contention Scheduling (Schwartz 2006).

In reality, the challenges to Contention Scheduling above do not address the model specifically. Rather, they present a debate about theoretical approaches. This debate, between IAC and recurrent connectionism, has been continuous since the 1980s (Cooper & Shallice 2006), and still has no definitive answer within the field of Computational Psychology. Consequently, the decision regarding whether to work off of Cooper and Shallice (2000) or Botvinik and Plaut (2003) should not be decided by a preference for either IAC or recurrent connectionism.

Instead, let us return to looking at the matter of applying learning mechanisms to a computational model of routine tasks in order to explore sequential learning in the brain. In this context, both models have deficiencies. The recurrent connectionist model of Botvinik and Plaut (2003) possesses a method for learning, but it requires a large number of training cycles. Consequently, this model's learning procedure does not accurately simulate a human's time-scale for learning (Botvinik 2008). On the other hand, Contention Scheduling does not have a training method specified at all in its

original form. Rather, it assumes that connections between nodes are already learned, and focuses more on the weighted inputs to schemas (Andronache & Scheutz 2002).

Because learning had not yet been implemented in Contention Scheduling, and because the recurrence network model had already explored and encountered trouble with learning, this paper explores sequential learning through Contention Scheduling. It is a successful IAC model of routine tasks, and has more flexibility to explore the role of learning in the brain because it has not been implemented with learning previously.

Comment [FI2]: Why?

Background Research on Procedural Learning

We have now established that Contention Scheduling has the most potential for simulating procedural learning. Now, let us explore what research has been done on learning within both Cognitive Psychology and Neuroscience, so that we can find the best way to apply learning to the model.

Contention Scheduling examines an area of procedural knowledge somewhere in between "motor response schemas" (Schmidt 1975), which are very specific motor response representations, and "scripts" (Schank & Ableson, 1977), which are higher level representations, such as chores or errands. This intermediate, distinct form of sequencing has not been studied as thoroughly in Cognitive Psychology. Many studies in Cognitive Psychology have made use of the Serial Reaction Task to evaluate sequence learning (Curran 1995), but this does not measure the sort of task learning explored in Contention Scheduling. Most of the research done on schema representation has been done in Computational Psychology (Cooper & Shallice 2006) which, as mentioned earlier, is lacking in results that examine learning.

Because of this, we will use research on the brain bases of learning from the neuroscience community to help implement learning within Contention Scheduling. Since the 1990s, most of the research that has examined plasticity in the brain has revealed Hebbian mechanisms for learning (Kolb 2003). However, this is not the only proposed mechanism for learning in the brain. O'Reilly and Rudy (2000) explore the pros and cons of Hebbian Learning as well as Error-Driven learning, in which the brain learns through its attempts to minimize errors. While Error-Driven learning has been thought to be implausible in the past, O'Reilly (1996) proposed an algorithm that might indeed be biologically possible. However, Gureckis and Love (2010) demonstrate that simple associative connections are more representative of the pattern of human learning results. This supports the use of Hebbian learning as opposed to Error-Driven learning. As a result, it appears that applying the concepts of Hebbian learning to Contention Scheduling would be the best way to simulate routine task learning in the brain.

The Contention Scheduling Model

Contention Scheduling is a neural network paradigm whose main purpose is to explore action selection in routine tasks. Before we address how we made use of the above ideas to change Contention Scheduling, let us first describe how the model works in its original form. After discussing the model in depth, we can then go into the changes that allowed it to account for learning and action correction.

Node Networks

Contention Scheduling uses a feed-forward neural network to represent action selection. There are three networks that interact with each other in this model – the resource, object and schema networks (Cooper & Shallice 2000). Each network contains nodes that receive and give off activation. However, each network has different specifications for how to weight different types of input to produce the final activation level for nodes of that network at each cycle of the model.

The Schema network is the central component of the Contention Scheduling. Each schema node represents some sort of goal that an agent wishes to achieve. The schema nodes are hierarchically organized, such that high-level, abstract nodes pass down activation to low-level, object-specific nodes. The connections between these nodes, as it is with all nodes in this model, are one-directional.

The Object network contains nodes that represent specific actions that an agent can perform. These actions can be abstract, and interact with high-level schemas, or they can be concrete, and interact with low-level schemas. Object nodes do not pass activation on to other object nodes, but rather receive activation from schemas and pass activation on to schemas. An object cannot both pass on and receive activation from the same schema – this is part of the feed-forward nature of the model.

The Resource network specifies what an agent has at their disposal to perform tasks. Resources can be specific objects that exist in the real world, as well as things like the attention of the agent. Like object nodes, resource nodes receive and pass activation to schema nodes, but not to each other, and not to objects. See Figure 1 for a visual representation of the communication across networks.

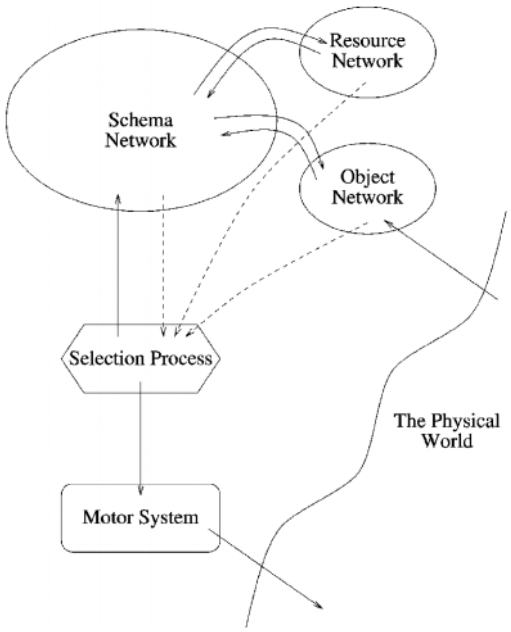


Figure 1. The fundamental components of Contention Scheduling, taken from Cooper and Shallice (2000).

Beyond the three networks described above, Contention Scheduling specifies two other parts of action selection. The first of these is the Supervisory System. This is a higher-level system that oversees the completion of a routine task. The way it most often interacts with the schema network is by passing down activation to the highest-level schema node that sparks the network to complete a routine task. The second part is Environmental Influence. This refers to what is in the real world. Unlike resource nodes, which usually specify the tools one uses to complete a task, environmental influences refer to objects in the real world that are part of the task at hand, orientation of objects, capacities of the agent in its

environment, etc.

Let us look at how these components interact with each other through an example: coffee making. This will be the example to which we return throughout this paper, and is also the simulation through which this model was tested. Cooper and Shallice (2000) used coffee making to run their original model, and the structure of the model is detailed in Figure 2. What Figure 2 lays out is the hierarchical structure of schemas, and how the objects interact with them. All nodes in italics are schemas, and all nodes in bold are objects. What are not present in the diagram, but are nevertheless important, are resources and environmental/supervisory influences.

Schemas pass down internal activation to other schema nodes (something that is also not depicted in Figure 2). However, they only do this if they are selected, which means they must have the highest activation at their hierarchical level and must be unachieved. For instance, say that *Coffee From Jar* is selected. That node will then pass down activation to all of the schema nodes that are its subgoals. These nodes include *hold* and *pour*, among others. Schemas also pass down activation to object nodes to which they are connected, but once again, only when they are selected. For documentation on the mathematical equations governing input activation, see the appendix of Cooper and Shallice (2000).

Schemas laterally inhibit each other as well. If a schema shares subgoals with another schema, then it will inhibit that schema. This process of competition allows a selected schema to pass on more inhibition to other nodes than it receives. Consequently, if we refer back to Figure 2, as *Coffee from Jar* becomes more active, it will pass on more

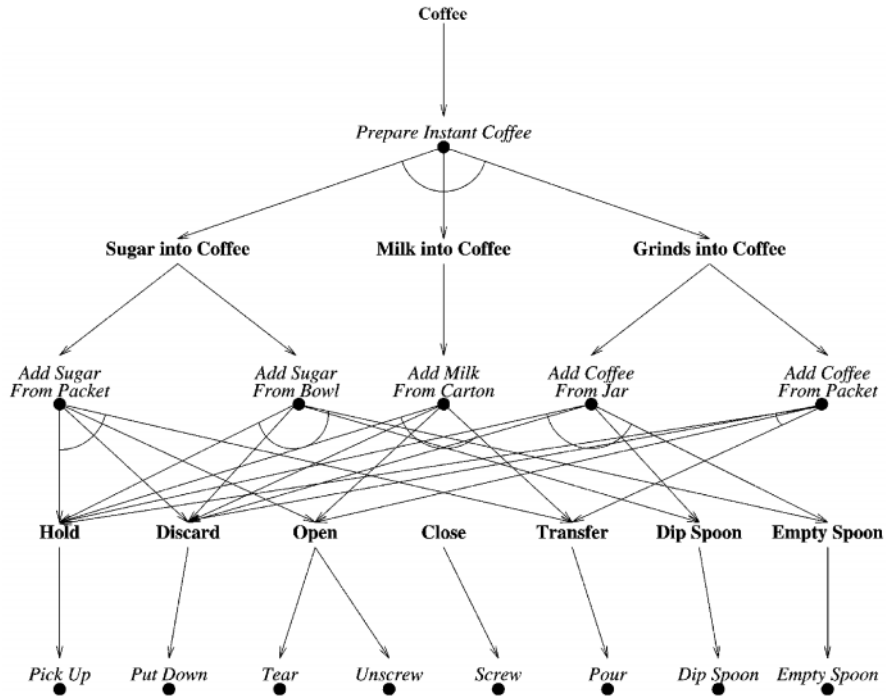


Figure 2. Hierarchical organization of schemas and objects, taken from Cooper and Shallice (2000). Schemas are in italics, and objects are in bold. Resources are not represented in this picture.

inhibition to nodes such as *Coffee from Packet* and *Milk from Carton,* which will further allow *Coffee from Jar* to become more active. Competing with lateral inhibition is a schema's self-activation. If a schema has not been achieved, it will pass on excitatory activation to itself. Once it has been achieved, it will inhibit itself.

Object activation is different from schema activation. Objects can only pass activation on to schemas, and can only receive from schemas. They will pass on activation no matter what – there is no dependency on selection or achievement, because they are not goals one must achieve. If two objects are used for the same function, they

will also laterally inhibit each other. For instance, when the *Prepare Instant Coffee* schema is activated, it will activate **Grinds into Coffee** as well as **Milk into Coffee**. However, both of these objects are used to fulfill one goal, and so they will inhibit each other. Object's self-influence is always positive, because there is no concept of achievement involved.

Resource nodes are even simpler than objects. They only give off activation to schemas, and only receive it from them. They do not have self-influence or lateral inhibition (i.e. they are not in competition with each other). As mentioned before, the only two resource nodes for coffee making are one's hands. Looking at hands as an example, they only pass/receive activation from the lowest level schemas, since they are only used for very basic goals. Even though the resources within coffee making are only one's hands, resources can be more diverse in nature within other tasks. For instance, attention could be considered a resource. Perhaps it could keep an overarching goal in mind (i.e. *Prepare Instant Coffee*), passing on a small amount of activation to that schema, while most of its activation is devoted to more specific schemas (such as *Pour*).

Environmental influences within the context of coffee making are relatively straightforward. If I am making coffee, and I have a jar of coffee grinds in my house, as opposed to a packet of coffee grinds, then environmental influences will pass activation on to the *Coffee from Jar* node, making it more likely to be selected than *Coffee from Packet*.

Parameters

While the core structure of the model is detailed above, the source of incorrect action selection within Contention Scheduling generally comes from its parameters. As described above, the type of activation passed between/within networks differs depending on the network in question. For instance, the calculations involved in determining the activation passed from schema to schema differs from those involved in determining the activation passed from objects to schemas. Consequently, adding these together requires parameters to determine the weight that each type of activation gets in determining the total activation into a given node. By adjusting these parameters, one is able to adjust how nodes influence each other, and consequently observe errors in the model's performance.

The first parameter is the Self:Lateral parameter. This adds together self-activation and lateral inhibition, which together yield the total "competitive" input to a node. Depending on the value of the parameter, a node might be more influenced by self-activation, or lateral inhibition. A value of 0.5 yields an even combination of the two types of input. However, the parameter can range anywhere from 0.5-0.65 and still lead to stable results (Cooper & Shallice, 1997).

Next is the Internal:External parameter. This adds together internal activation (from schema to schema) and external activation (from one network to another), giving the total "noncompetitive" input to a node. The parameter's value differs between objects and schemas – a stable range for objects is between 0.06 and 0.14, heavily favoring external activation, because there is no top-down internal activation of objects. Schemas, on the other hand, have a stable range between 0.94 and 0.96, heavily favoring schema activation over external influences (Cooper & Shallice, 1997). However, the parameter

can be lowered in value within the realm of schemas to simulate the errors observed in patients with Action Disorganization Syndrome (Cooper & Shallice, 2000).

Once the total competitive and noncompetitive activations have been determined, they can be combined to give the total novel input to a node using the Competitive:Noncompetitive parameter. A stable range for this value is between 0.5 and 0.65. This activation value is then used in the node's update function, which determines the activation of the node at time t+1, given its activation at time t and total new input. This function, rather than weighted addition like the previous parameters, uses a sigmoid approximation to make sure that the total activation of a node is between 1 and -1.

There are two parameters that are part of the equation for total activation. The first one is Persistence, which influences the decay of the node's activation over time. A stable range for Persistence is between 0.79 and 0.81. The second value is noise. Noise is used to make sure that nodes receiving similar input still have at least slightly different activations, which is more representative of how activation is passed in the brain. Most of the time, noise randomly affects activation to the degree of 0.001, but increasing noise creates more incorrect action selection.

The parameters mentioned above determine the total activation of a given node after each cycle of the model. However, there are two more parameters that are necessary for the model to function. First is the selection threshold – a schema is deemed selected by the model if its activation exceeds the selection threshold and it is the highest activated schema at its level (i.e. two nodes may be activated enough, but only one can be selected). The threshold is usually set to 0.6, but can stably exist between 0.5 and 0.95 (Cooper & Shallice, 1997). The second parameter is the resting activation of nodes. This

value is meant to simulate neural activity – neurons at rest are still slightly active. Consequently, when nodes in the model have not received input, they will return to a resting activation level. This model sets that level at 0.10 for the most part, but the model can exist stably anywhere in between 0.09-0.11.

Lastly, Cooper and Shallice (2000) discussed in detail the different types of errors that would occur based on manipulating the parameters of the model. The three most prevalent ones, and the ones that we will examine closely in this model, are omission, sequence and capture. Omission errors occur when one forgets a step in performing an action, sequence errors occur when one performs the steps of a task out of order, and capture errors occur when one does the wrong action in the place of the correct one (i.e. getting coffee from a packet when one only has a jar from which to retrieve coffee). For further information on these errors, see Cooper and Shallice (2000).

Changes to the Model

How Hebbian Learning Applies to Contention Scheduling

As mentioned earlier, one of the biggest problems with Contention Scheduling is that it does not have a proper learning mechanism (Andronache & Scheutz 2002). Instead, it explores the nature of how nodes pass activation to each other, and assumes that the weights of the connections between different nodes are fully learned. This version of the model does not adjust the nature of the input/output mappings described by Contention Scheduling. Instead, it adds weights to them, such that the amount of

previously determined activation that one node passes to another is affected by the strength of the connection between those two nodes.

Let us consider a more concrete example of this by revisiting the description of Contention Scheduling above, looking specifically at the effects of the *Prepare Instant Coffee* schema on the *Coffee from Jar* schema. In the original model of Contention Scheduling, the connection between these two nodes was fully developed, and therefore *Prepare Instant Coffee* passed down all of its activation to *Coffee from Jar*. But how would this sequence of actions be different if this was the first time that someone was learning to make instant coffee? We can suppose that, because this is the first time someone is learning to make coffee, *Prepare Instant Coffee* and *Coffee from Jar* are not well-associated with each other, such that little of the activation from one affects the other. However, we assume that each individual schema/object is something that the learner already has a representation of, as concluded by Dixon (1982) is necessary for learning new procedures.

Within this model, Hebbian connections among weights are dynamic. During each cycle through the simulation, Hebbian connections are updated by coinciding activation between nodes for all connections within the model. The type of Hebbian update used here is based on an online learning paradigm utilized by Eberhard, Scheutz and Heilman (2005), which prevents connections from ever growing beyond a full weight of 1.0, which, for the purposes of this paper, is the weight of a completely learned connection. The equation for this paradigm is

$$\Delta act/\Delta t = netin - act \bullet (netin + decay). \quad (1)$$

Additional Changes to the Original Model

In their 2002 paper, Andronache & Scheutz outlined some other issues with the original Contention Scheduling model. Besides a lack of learning, they showed that while the model does exhibit errors in action selection seen in Action Disorganization Syndrome, it does not address mistakes after they are made. Botvinik and Plaut's (2003) model does not address this problem either. For instance, say that the *Coffee from Jar* node is selected within the model - when it passes down activation to its subschemas, the *Close* schema becomes selected before the *Open* schema, and the *Open* schema is simply skipped by the model, leading to an Omission error. In the original model, this error would stay in place, and the model would continue working for as long as it could, but never would complete the *Coffee from Jar* goal because of the omission error. This problem, the problem of action correction, is another aspect of the model explored in this paper.

Action Correction is addressed here through the use of bottom-up connections within the hierarchy of Contention Scheduling. The goal was to use bottom-up connections to implicitly communicate the completion/incompletion of certain lower level goals to higher-level schemas. The idea that bottom-up and top-down connections work together to balance each other has been supported in a number of experiments, particularly those that look at vision (Reviewed by Wolfe, Butcher, Lee & Hyle 2003). While vision is a different type of cognitive process than that explored here, such a strategy is nevertheless cognitively supported, and worth investigating.

For this model, bottom-up connections were implemented in schema nodes only. Each schema node connects to the object nodes that externally activate it, as well as to the higher-level schema nodes that internally activate it. Schemas use bottom-up connections by passing external activation on to their respective higher-level schemas and objects as described in the Parameters section above. External activation was used as opposed to internal in order to have the bottom-up connections balance out with the internal connections from higher-level schemas to lower level schemas (because all top-down connections are internal). We decided to stay within the framework of the model, instead of introducing a new "action correction" network, in order to measure the ability of the model to implicitly correct itself.

Based on the information reviewed here, the most efficient and cognitively supported method for exploring learning in Contention Scheduling is through creating Hebbian connections among nodes. As well, adding bottom-up connections to the model is a way to introduce error recovery to the model implicitly, and without changing its structure greatly. With these two changes, the model maintains its hierarchical structure, while exploring more cognitive phenomena.

Methods

Implementation Methodology

We returned to the paradigm used by Cooper & Shallice (2000) to explore the roles of learning and action correction within Contention Scheduling – Coffee making. This simulation environment consisted of 25 schema and object nodes, as well as two

resources nodes (one for each of left and right hands). To get a sense of how activation flows between nodes, refer to the background section above on Contention Scheduling.

The platform used for implementing the model was Matlab. Each type of activation/inhibition for a given node was represented in an array – for instance, there was one array that stored the lateral inhibition for each node, and another that stored the external activation for each node, etc. These arrays were then added cell-wise to each other according to the parameters of the model. The weighted connections between the nodes were stored in a 25-by-25 matrix, and were included in all activation calculations. This is different from the original model, in which all connections were assumed to be valued at 1. Each cycle through the model simulation included an update function for the weights, using the Online Learning Paradigm (Eberhard, Scheutz and Heilman 2005) from equation 1.

The model simulation took the form of a Matlab function script. One run through the script reflected the performance of an entire procedure. The function would run for as long as was necessary to complete the task, but would stop at 550 cycles if the model had not yet been completed. Completion of the task was measured by assessing whether the highest-level schema, *Prepare Instant Coffee*, was completed. The model would stop running when an error was detected (i.e. omission, addition or sequence), except in action correction simulations.

Simulating the Original Model Using Matlab

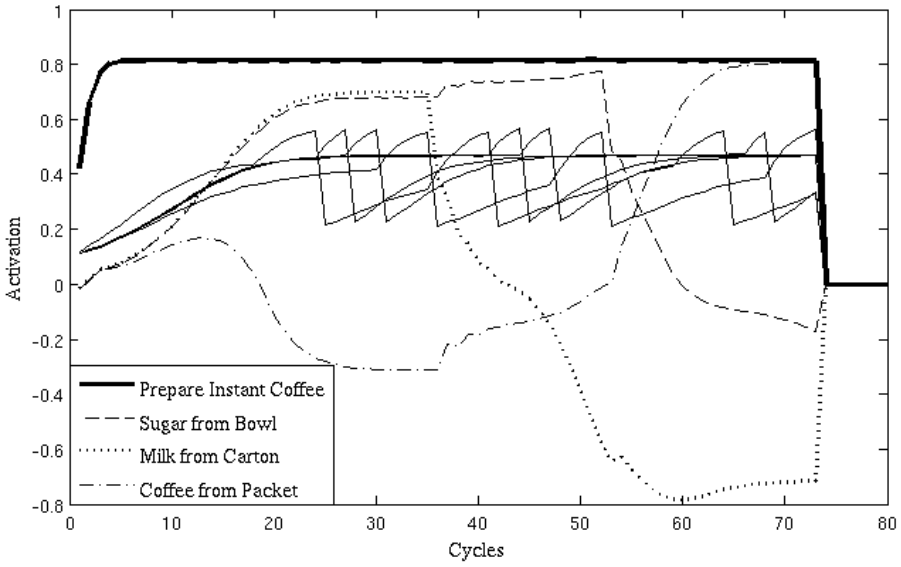
Matlab was not the original interface for simulating Contention Scheduling. Consequently, it was important to make sure that, even with a different simulation tool,

the model would still function as it did originally for Cooper and Shallice. In order to make sure it ran properly, we first simulated the original model, without Hebbian Learning, to make sure it replicated earlier findings.

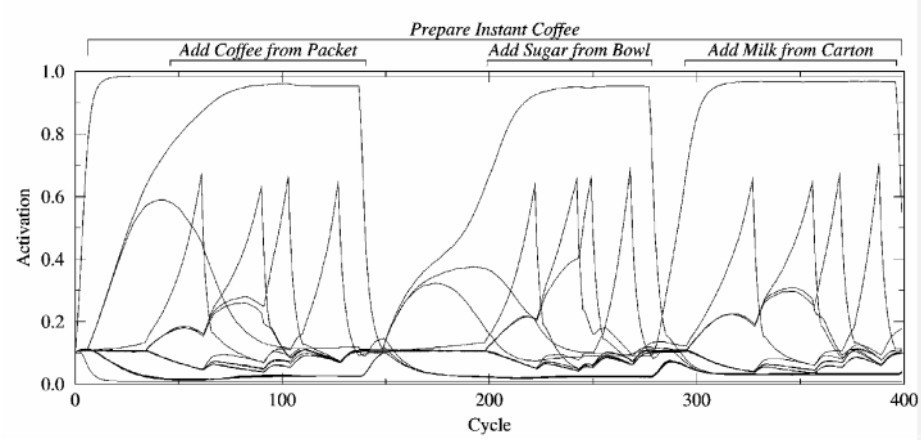
There were a few details of the implementation that caused differences in exact replication of the data, but these were a consequence of the Matlab platform, rather than of incorrect implementation. The model still accurately simulated Contention Scheduling with regards to activation and error production. One difference between the two is in the amount of cycles needed to complete the task: for Contention Scheduling, originally the necessary amount of cycles was ~600, whereas for the Matlab implementation, the necessary amount of cycles was around one tenth of that (i.e. 60 cycles). The reason for this was because of the inherent differences in the program – resource allocation and environmental influence – had to be implemented differently. Yet the model still produced the same outcomes. Figures 3 and 4 show simulated activation levels for schemas in the original model as well as in the Matlab implementation. Though there are a few slight differences, the overall activation flow is the same for both.

For larger values of the Internal:External parameter, the model implemented here produced little to no errors. Similarly, as the value of the parameter was decreased, it produced increasing amounts of omission and sequence errors, which mirrors the patterns produced by Cooper & Shallice. Consequently, the Matlab implementation of Contention Scheduling can be manipulated here, and we can assume that the manipulations are valid with respect to the original model.

Trial Simulations With Model Variations



*Figures 3 & 4. (*Above) Activation levels for schema nodes by cycle within the Matlab simulation of Contention Scheduling. (Below) Activation levels for schema nodes by cycle within the original model, taken from Cooper and Shallice (2000).



Using this basic function script, we ran multiple trials, each time modifying certain aspects of the model. The first type of trial was one in which the model learned how to make coffee as if it had never done so before – all of the weighted connections between nodes were equal to 0. In order to simulate how a person learns a new procedure such as coffee making, the internal activation function was altered such that it relied more heavily on the supervisory system. Instead of having internal activation come from the higher level nodes only, the supervisory system would pass direct internal activation down to higher-level and lower-level nodes simultaneously, both binding them together and completing the task in the correct order. This was meant to simulate how a person might follow explicit directions in what John R. Anderson (1982) calls the "Declarative Stage", or the first stage in procedural learning. It also reflects the difference between one's explicit and implicit knowledge of a procedure, which was demonstrated by Doya et al. (1999) in their study on neural networks in learning sequential procedures.

The second type of trial simulated incorporating a new aspect of coffee making into an already learned model. For example, let us examine a person who has always learned to make coffee using *Coffee from Jar*, and has never been faced with using *Coffee from Packet* before. In this trial, we assume that the model already has a fully developed concept of making coffee, but that it was asked to change its usual implementation in order to take coffee from a packet, instead of a jar. Consequently, whereas all other connections are ~1, the connections to and from the *Coffee from Packet* node would be 0.

In order to simulate combining explicit learning with an implicit, learned task, a combination of supervisory activation and regular internal activation was used. The new schema was learned using supervisory activation, and following the completion of that

new schema, the model reverted back to implicitly passing internal activation from one node to the next. The implicit form of activation that comes from a learned model can be compared to John R. Anderson's (1982) "Procedural Stage", in which the learner has a conceptualization of the procedure to perform.

Within all trials, the Internal:External parameter was manipulated to compare to the results found by Cooper and Shallice (2000) when they inspected Action Disorganization Syndrome through Contention Scheduling. As well, in order to differentiate observed changes in each individually, all trials were repeated twice – once with action correction, and once without.

Results

The models were analyzed using two different types of data – the amount of cycles needed for completion, as well as the amount of errors observed over a large amount of simulations. Within the data on errors, we differentiated between three different types of errors: omission, sequence and capture. These were the three errors most linked to Action Disorganization Syndrome in the Cooper and Shallice (2000) study, and they were also the most frequently observed errors within that study.

Learning vs. Control

In order to observe the effects of learning, we compared the data on cycles and errors between learning trials and the control run of the original model. Our hypothesis was that implementing Hebbian Learning within the model would allow for the model to

successfully complete, while demonstrating behavior reminiscent of a person learning a new task. What was most important in this comparison was that, when the learning models worked to completion, they would do so at a slower rate (i.e. over more cycles) than the original model.

Data for learning simulations came in two forms: the first form was for coffee making as a completely unlearned task. The second was for coffee making in which all parts of the model had been learned, except for one (i.e. it was learning to do *Coffee from Packet* but had already learned to integrate *Milk from Carton* and *Sugar from Bowl* into the model).

The first piece of data recorded was error patterns. The original hypothesis was that there would not be a large difference in error patterns between learned and unlearned simulations, since neither type of simulation had a way to correct or measure errors within the simulation run. In general, this hypothesis held – however, as is evident in Figure 6, there were lower levels of overall error for completely unlearned simulations, whereas partially learned simulations exhibited somewhat similar behavior to control simulations. While this effect was not expected, explanations as to why it occurred will be discussed later. We will also explain the brief separation between Partially Learned and Learned simulations seen for Internal:External = 0.75-0.85 observed in Figure 5.

If we isolate the different types of errors, and instead view the effects of learning individually, the results are a little bit different. Here, data for omission, sequence and capture errors were recorded individually. For omission, one of the most prevalent errors within the model, there was no significant effect between learned and partially learned models. However, for completely unlearned models, there was a slight difference in the

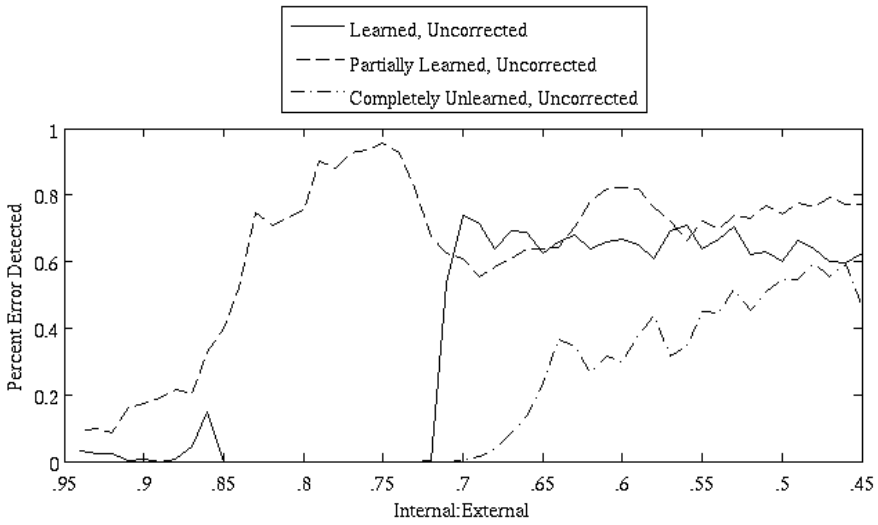


Figure 5. Percentage of simulations reporting errors for Learned, Partially Learned and Unlearned simulations. Simulations were conducted for each group by manipulating the Internal:External variable.

percentage of errors observed as compared to learned simulations ($F = 37.13$, $p = .00$). Figure 6 gives a visual account of these findings. Looking at sequence errors, there was no significant effect of learning on the amount registered. However, for capture errors, there was a significant increase in the amount registered for a partially learned sequence compared to a completely unlearned one or a learned one ($p < .05$). Another important aspect of measuring learning models was the amount of cycles needed for completion. The original hypothesis was that it would take more cycles to complete a learning simulation than a learned simulation. This hypothesis was also supported by the results of

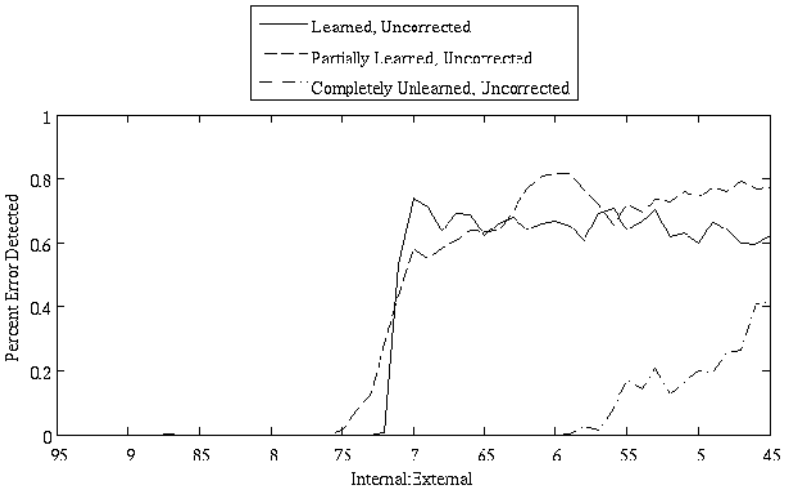


Figure 6. Percentage of omission errors detected in learned, partially learned and completely unlearned simulations. None of the simulations included action correction.

the study. Accounting for errors that caused the models to not complete at all, the amount of cycles necessary to complete a learned model were fewer than for a completely unlearned model ($p = .01$), supporting the original hypothesis. The average amount of cycles for completion for a learned model was 77.94 ($SD = 11.46$), whereas the average amount of cycles necessary for completion for a completely unlearned simulation was 122.52 ($SD = 1.27$). However, there was no effect on cycles found between learned and partially learned simulations.

Lastly, the learning algorithm allowed the model to learn the task within a reasonable time-scale. After two consecutive simulations of the model with a learning rate of 0.04 (Eberhard, Scheutz & Heilman 2005), the average weights between nodes

was 0.74 (SD = 0.16), which was a large enough value for the model to then make coffee without the help of the supervisory system. This supports the hypothesis that Hebbian learning is an effective tool for developing connections within the model.

Overall, most of the hypotheses regarding learning were supported. The error percentage between groups was, for the most part, similar, and the amount of cycles needed to complete a completely unlearned task was significantly larger than for a learned task. In general, the completely unlearned models exhibited behavior more closely tied to our hypotheses than partially learned models.

Action Correction vs. Control

For action correction, the hypotheses differed in many ways from learning. First, we predicted that there would be fewer errors in corrected models. Second, we predicted that there would be no significant effect of action correction on the amount of cycles needed for completion.

To assess the effects of action correction, let us once again look at errors individually, instead of errors as a whole. First, for omission errors, there was a significant main effect for action correction on the percentage of omission errors detected ($F = 56.69$, $p = .00$). Figure 7 demonstrates the difference between the original model and the corrected model with regards to omission errors.

On the other hand, the measurements of sequence errors used here increased as a consequence of action correction. Even though these measurements registered more sequence errors, there was not, most likely, an increase in actual sequence errors as a result of adding in action correction. This event is further explained in the discussion

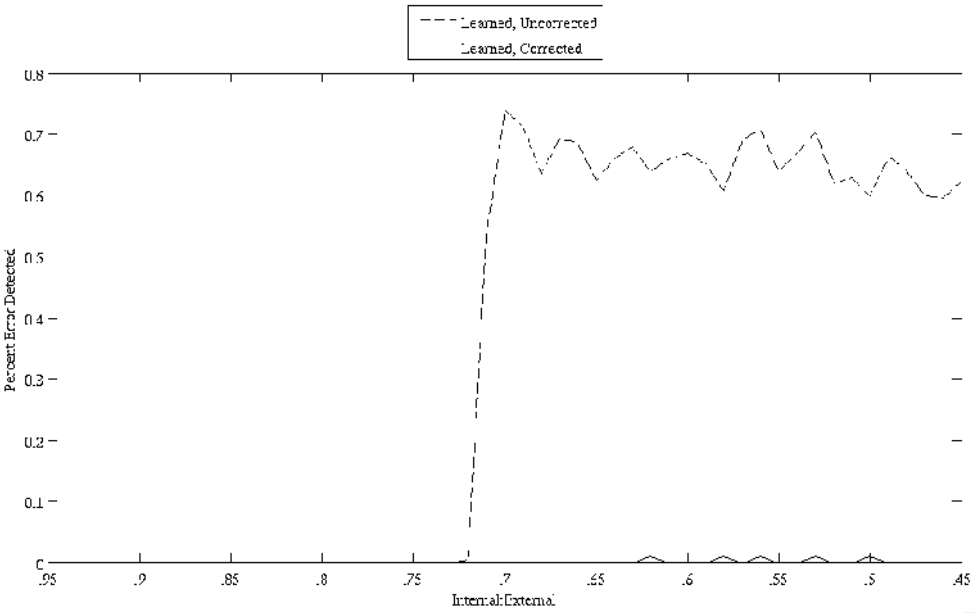


Figure 7. Percentage of omission errors detected for corrected and uncorrected simulations of the model. Both types of simulation were learned.

section. As for capture errors, there was no significant effect of action correction on the percentage recorded, which supports the null hypothesis.

Interaction of Action Correction and Learning

As mentioned earlier, we predicted no interaction effects for learning and action correction. In general, there were no interaction effects, even though there were main effects for both variables. However, there was one interaction tendency observed, though it was not statistically significant ($p = .12$). As Figure 8 demonstrates, it appeared that, for Internal:External values between close to 0.90 (i.e. 0.85-0.95), the interaction between learning and action correction caused an increase in the average amount of cycles needed

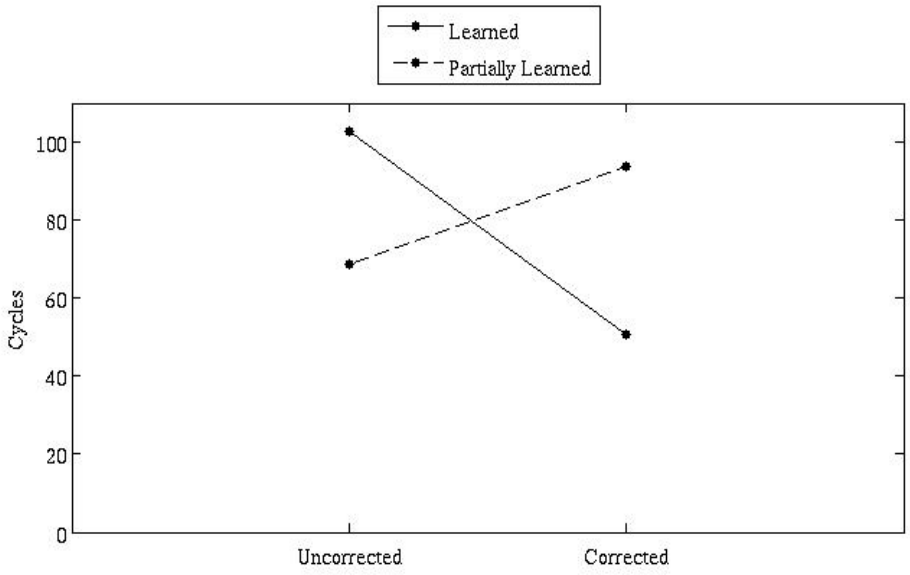


Figure 8. Interaction effects observed over cycles needed for completion for Learned and Partially Learned simulations. This interaction was not significant ($p > .05$).

for completion for Partially Learned simulations, whereas it caused a decrease in the average amount of cycles needed to complete Learned models. This finding was not a part of the original hypothesis. It is worth noting as well that no interaction effect was observed between Learned and Completely Unlearned models.

Discussion

Overall, the changes made here to Contention Scheduling have addressed many of the issues raised with the model. The addition of Hebbian Learning reflects learning in humans and has been integrated into the model in a way that does not affect action

selection, as originally hypothesized by Cooper and Shallice (2000). Not only is the model able to learn a task as if it had no knowledge of how to perform it before, but it is also able to incorporate a new task into a learned network. On top of that, the incorporation of our action correction component, while not perfect, made significant progress towards establishing a fully self-corrective model.

Let us explore some of the unexpected results observed in the simulations. First of all, there was a higher percentage of capture errors observed in partially learned models that were not corrected. This seems plausible if we are to consider a human who is learning to incorporate a new action into a learned sequence. If a person stops paying close attention to the sequential procedure, and there is a higher implicit tendency to perform a previously learned action (because of the higher weight values), then it is likely that those strong connections will dominate, even in the face of environmental influences. Because this tendency was observed in high values of Internal:External, we know that internal activation was a much stronger presence than external activation. Consequently, the fact that incorporating a new action into a learned sequence leads to more capture errors is a likely consequence of the situation.

The second finding that was not predicted was the increase in sequence errors observed in corrected simulations. This event highlights a limitation of the current implementation of the model. For this model, if an omission error was detected, the model would correct itself by going back and performing the omitted action. However, because there is no way to "undo" a previously completed action, the model would still emit a sequence error, because the omitted action was still performed after the completion of a sequentially later action. If the model incorporated a way to look at "undoing" wrong

actions, it would have solved this issue.

The third finding demonstrates another limitation within this model – the inability of the model to account for capture and sequence errors. As mentioned above, the action correction implementation was not perfect. It successfully dealt with repetition errors, because it always worked to completion. It also successfully dealt with omission errors. However, even though capture errors had a low prevalence in general, and had a lower prevalence in corrected models than in uncorrected ones, they were not successfully eliminated when they occurred. Including the "undo" aspect of the model could potentially solve this problem, but it would also require the model to have an awareness of the fact that the resources for one schema are present when the resources for another are not.

Finally, though the interaction effect of action correction with partially learned models was not significant, it is still worth considering, especially for future simulations. One possible explanation for the increase in cycles needed for Partially Learned models is the fact that, for observed omission errors, the model had to go back and complete previously omitted tasks, requiring it to take more time to finish. On the other hand, action correction would have caused Learned models to complete more quickly because they were receiving bottom-up activation in addition to top-down activation, causing them to reach threshold much more quickly.

Though there were some limitations to this study, as demonstrated by a few of the unexpected results, there were also strengths. First of all, Hebbian learning seems to be both the easiest and most efficient learning method for the model. It successfully simulated many observed human tendencies (with regard to cycles necessary for

completion, for instance), and taught the model in a reasonable amount of time.

Second, while the action correction implementation has a lot of room left for future exploration, the way in which it was implemented reflected cognitive research well. The interaction of top-down and bottom-up connections has been observed, as mentioned earlier, in many cognitive processes, and there is thorough evidence for it in the brain. Consequently, the fact that this form of action correction was successful suggests that this version of the model reflects cognitive performance.

Lastly, having learning be driven by activation from the supervisory system was both effective and supported the neuroscience. As Hikosaka et al. (1999) demonstrate, the structure of a routine task in the brain when it is first learned is different from its structure when it is recalled once it has been learned – there is more influence from the executive in a novel task.

The limitations of this study do not highlight issues with the new concepts presented in this paper. Rather, they point out future areas in which to explore Contention Scheduling more closely. As well, the strengths of this model show that, even though it is still not complete, incorporating Hebbian weights and bottom-up connections move the model in towards a more full and thorough computer model of cognitive functioning.

Future Directions

There are further improvements that can be made to this model to make it even more reminiscent cognitive functioning. As mentioned in the discussion section, there is more that can be explored within the domain of action correction. Bottom-up connections

seem to mediate errors well, but they are not the final solution to the problem of action correction. One important aspect of action correction that needs to be explored further is undoing errors. One potential way to approach this issue is to create an "undo" action that could interact with the system in much the same way as the supervisory system.

With a model like this that closely represents cognitive function, there are further applications within cognitive science. Using this model as a tool, we could subject it to parameter changes that would simulate situations that are not easily controlled within a lab setting, and see how they behave. This could further our understanding of sequential learning in the brain.

Finally, this model can be applied outside of cognitive science to the area of behavior-based robotics. It could become a tool for robots to store procedures hierarchically, and to learn new ones dynamically, without having to be programmed to do so. This would be a powerful tool for allowing robots to operate automatically, without much human oversight.

Conclusion

Comment [FI3]: In your conclusion, be sure to restate explicitly what your project has found/contributed to the field.

Subject to the new additions of learning and action correction, Contention Scheduling maintains its connections to cognitive phenomena. The addition of action correction significantly affected the amount of errors recorded by the model for some types of errors, but not all. The addition of learning affected the time-scale of the model, and took a reasonable amount of time to train the connections between nodes. The model exhibited some interesting and unexpected behaviors as a consequence of the changes

made to it, but those results did not contradict its demonstrated applications to cognition – instead, they pointed out future changes that can be made to the model to increase its ability to realistically simulate human behavior while maintaining its fundamental structure. Consequently, the addition of action correction and learning discussed here furthers our ability to model routine tasks in humans more thoroughly. With further development, Contention Scheduling has the potential to be one of the first computational models to identically replicate human behavior, as well as act as a paradigm for learning within behavior-based robotics.

References

(No Author). Changing Your Mind: On the Contributions of Top-Down and Bottom-Up Guidance in Visual Search for Feature Singletons, *Journal of Experimental Psychology: Human Perception and Performance*, 29(2), 483–502.

Anderson, J. R. (1982). Acquisition of cognitive skill. *Psychological Review, 89*(4), 369-406. Retrieved from http://act-r.psy.cmu.edu/papers/63/ACS_JRA_PR.1982.pdf

Andronache, V., & Scheutz, M. (2002). *Contention scheduling: A viable action-selection mechanism for robotics?* Unpublished manuscript.

Botvinick, M., & Plaut, D. C. (2003). Doing without schema hierarchies: A recurrent connectionist approach to normal and impaired routine sequential action. *Psychological Review*, 111, 395-429.

Botvinick, M. M. (2008). Hierarchical models of behavior and prefrontal function. (Invited review). *Trends in Cognitive Sciences*, 12, 201-208.

Botvinick, Matthew M., & Plaut, David C. (2006). Such stuff as habits are made on: A reply to Cooper and Shallice. *Psychological Review*, 113(4), Oct 2006, 917-927. doi: 10.1037/0033-295X.113.4.917

Cooper, R., and Shallice, T. (2000). Contention scheduling and the control of routine activities. *Cognitive Neuropsychology* 17(4), 298–338.

Cooper, R. P., & Shallice, T. (2006). Hierarchical schemas and goals in the control of sequential behavior. *Psychological Review*, 113, 887–916.

Cooper, R., & Shallice, T. (1997). Modeling the selection of routine actions: Exploring the criticality of parameter values. *Proceedings of the Nineteenth Annual Conference of the Cognitive Science Society,* Stanford, CA.

Cooper, R., Shallice, T., & Farringdon, J. (1995). Symbolic and continuous processes in the automatic selection of actions. In J. Hallam (Ed.), *Hybrid problems, hybrid solutions* (pp. 27) IOS Press.

Dixon, P. (1982). Plans and written directions for complex tasks. *Journal of Verbal Learning and Verbal Behavior,* 21, 70-84.

Doya K. (1999). Multiple representation and algorithms for sequence learning. *2nd International Conference on Cognitive Science*, Tokyo, 17-19.

Eberhard, K. M., Scheutz, M., & Heilman, M. (2005). *An empirical and computational test of linguistic relativity*. Unpublished manuscript.

Guthrie, J. T., Bennett, S., & Weber, S. (1991). Processing procedural documents: A cognitive model for following written directions. *Educational Psychology Review, 3*(3), 249. Retrieved from http://www.springerlink.com.ezproxy.library.tufts.edu/content/n125n36014lnr287/fulltext.pdf

Hikosaka, O., Nakahara, H., Rand, M. K., Sakai, K., Lu, X., Nakamura, K., . . . Doya, K. (1999). Parallel neural networks for learning sequential procedures. *Trends in Neurosciences, 22*(10), 464-471. doi:10.1016/S0166-2236(99)01439-3

Kolb, B. (2003). The impact of the hebbian learning rule on research in behavioural neuroscience. *Canadian Psychology/Psychologie Canadienne, 44*(1), 14-16. doi:10.1037/h0085813

Norman, D. A., & Shallice, T. (1986). Attention to action: Willed and automatic control of behavior. In R. J. Davidson, G. E. Schwartz, & D. Shapiro (Eds.), Consciousness and self-regulation: Advances in research and theory. New York:

Plenum Press.

O'Reilly, R.C. (1996). Biologically plausible error-driven learning using local activation differences: the generalized recirculation algorithm. *Neural Computation,* 8, 895–938.

O'Reilly, R. C., & Rudy, J. W. (2000). Computational principles of learning in the neocortex and hippocampus. *Hippocampus, 10*(4), 389-397. doi:10.1002/1098-1063(2000)10:4<389::AID-HIPO5>3.0.CO;2-P

Petersen, S. E., Van Mier, H., Fiez, J. A., & Raichle, M. E. (1998). The effects of practice on the functional anatomy of task performance. *Proceedings of the National Academy of Sciences of the United States of America, 95*(3), 853-860.

Rumelhart, D. E., & Norman, D. A. (1982). Simulating a skilled typist: A study of skilled cognitive-motor performance. *Cognitive Science*, 6, 1-36.

Schank, R.C., & Abelson, R. (1977). Scripts, plans, goals and understanding. Hove, UK: Lawrence Erlbaum Associates Ltd.

Schmidt, R.A. (1975). A schema theory of discrete motor skill learning. *Psychological Review*, 82(4), 225–260.

Schwartz, M. (2006). The cognitive neuropsychology of everyday action and planning. *Cognitive Neuropsychology, 23*(1), 202-221. doi:10.1080/02643290500202623