Lake:

A Cognitive Architecture for Menton Theory

By

Mackenzie Ostler

A thesis submitted to the Faculty of Cognitive Science in partial fulfillment of the Undergraduate requirements for the degree of

Bachelor of Cognitive Science

Carleton University

Ottawa, Ontario

Abstract

Davies and Fortney (2012) introduced the menton theory of boredom based on the allocation of mental resources. This paper will continue to theorize about their menton theory, as well as introduce Lake, a cognitive architecture that was built upon this theory. This paper examines the initial steps towards creating a comprehensive cognitive architecture of this theory and outlines areas where there is room for improvement, as well as how those improvements could be accomplished.

1. Introduction:

Cognitive modeling is an important aspect of building, communicating, testing and improving theories. However, as the audience of a theory grows, communicating and combining ideas can become more difficult, as the theory becomes more distant from the wider audiences' domains of expertise. Since cognitive science is such an enormous field, often two researchers in cognitive science will know very little about each other's domains. As a result, cognitive architectures are an important tool in bridging these gaps in communication (Anderson, 1993). Cognitive architectures not only help to visualize, manipulate and test theories of cognition, but they should also try and bridge all the various aspects of cognition to form an encompassing architecture. Of course, this could not nor should not be done all at once, and should be something that evolves over time. There are several existing successful examples of cognitive architectures, including but not limited to, ACT-R (Anderson & Lebiere, 1998), Python ACT-R (Stewart & West, 2007) and SOAR (Laird, 2012). This paper will focus on the first steps towards building Lake, a cognitive architecture built on the menton theory of cognition. Before introducing the menton theory, some context will be given by briefly reviewing other theories of mental resource.

1.1 Multi-Task Interference:

Multi-task interference is explained by several varying theories, one of which is the menton theory, and is the focus of this paper. Various tasks require varying amounts of attention as some task are inherently more compatible than others. For example, it seems easy to wash dishes and listen to the radio, but difficult to listen to the radio and read a book. Additionally, the same task might not always occupy the same amount of attention on various occasions. For example, if a person is eating alone at a restaurant, they would be more aware of

the conversations, sounds and general goings on in the restaurant surrounding them, compared to if they were at the restaurant with another person and engaged with them in a conversation. If this conversation were to come to a lull or become uninteresting, they might once again become more aware of their surroundings.

I will review a few theories that explain multi-task interference.

1.2 Central Bottleneck Theory:

The central bottleneck theory posits that all tasks must be processed via a central processing unit. When multitasking, these tasks are all processed by the central processing unit, causing a bottleneck, and therefore when multitasking there will always be some degree of interference (Meyer & Kieras, 1997). There is no anatomical evidence of the central processing unit. Furthermore, the theory fails to distinguish between what caliber of tasks require the preprocessing of the central-executive, and which can be processed locally (Pashler & Johnston, 1998).

1.3 Central Capacity Theory:

The central capacity theories of attention suggest that attention is a single resource that can be allocated to several tasks. Multitasking requires this resource to be divided between the tasks; the more tasks, or the more challenging the tasks, the fewer remaining resources to allocated to each task (Tombu & Jolicoeur, 2013). According to Kahneman's 1973 model, psychological arousal is the primary influencer in the amount of attention an individual has available to commit to any task(s). Therefore, if an individual completes the same task on two separate occasions, and on one occasion is consistently distracted by hunger, they may perform that task with less attention, and therefore with less success.

Single resource theories, including Kahneman's 1973 model, theorize that there is some singular mental resource that is used by all tasks (e.g., attention). However, singular resource models fail to explain why some pairs of tasks interfere more than others, since they rely on interference being related to number of tasks, rather than type of task (Navon & Gopher, 1979; Heuer, 1985). For example, reading and listening to music (both auditory) being more challenging than reading and drawing (auditory and visual). Following single resource theory, these two examples should be equally challenging, since both involve two tasks, however this is shown to not be the case. Two tasks of the same type have a higher interference than two tasks of differing types (Heuer, 1985), something which single resource theories fail to explain.

1.4 Multiple Resource Theory:

Another attempt to explain dual-task is multiple resource theory (Wickens, 2008). Multiple resource theory is a four-dimensional model of attention, where the dimensions represent distinct resources. The dimensions are: stages of processing, codes of processing, modalities and visual channels. The *stages of processing* dimension indicate that working memory uses different resources than those used by selection and execution of tasks. *Codes of Processing* dimension indicates that linguistic-based activities and spatial activities use different resources. The *modalities* dimension is nested within perception, and therefore does not manifest within cognition or response, rather it indicates that auditory perception is distinct from visual perception. *Visual Channels* distinguishes between focal and ambient vision (Wickens, 2008). This theory explains the why it is easier to draw and listen to music at the same time (visual and verbal) compared to reading and listening to music (verbal and verbal).

1.5 Menton Theory:

The menton theory, the theory behind Lake, proposes that the brain uses mentons like a mental currency, to execute tasks, determine the execution quality and whether the current set of goals/actions are stimulating enough to avoid boredom (Davies & Fortney, 2012). This theory draws from multi-task interference, central bottleneck theory and multiple resource theory to propose a new theory of mental resource. The central-executive allocates mentons the same way a planning committee allocates resources, based on priority, urgency and cost. The menton theory aligns with central capacity theories, by suggesting that the mentons are a single resource that is required for task execution. However, it diverges from single resource theories by suggesting there are different "pools" of mentons that belong uniquely to different areas in the brain. For example, there may be a pool of mentons reserved for motor control and a separate pool for the visual processing. These pools do not refresh quickly enough to avoid multi-task interference, and mentons belonging to one pool cannot move to another pool when there is a shortage or surplus (Davies & Fortney, 2012).

Different tasks require different quantities of mentons, and mentons originating from particular pools. If a task requires more mentons than the pool has left to allocate, the quality of the task or the number of tasks drawing from that pool must change (Davies & Fortney, 2012). For example, one can doodle and listen to music without much effort, since doodling requires very few mentons, leaving enough to satisfy the requirement of listening to music. However, if one was listening to music and then started doing a challenging math exercise, one might struggle to continue doing both tasks. This is because math requires more mentons than doodling, and by allocating more mentons to math, there are fewer mentons available to execute the task of listening to music. Davies and Fortney's (2012) theory goes on to explain not only what happens when there is a menton shortage, but also when there is a menton surplus. A

surplus of mentons, resulting from too many unused mentons, explains why an individual experiences boredom. By having too many inactive mentons, an individual's mind completely disengages from the task that is failing to occupy enough of their mentons. Andrade (2010) showed that students who were listening to a boring lecture recalled more facts if they doodled during the lecture. Students who were not permitted to doodle completely disengaged from the lecture, spending their time daydreaming or thinking about unrelated things. The menton theory explains this with menton surplus. The students who were allowed to doodle, occupied their surplus of mentons on the task of doodling, which in turn facilitated increased engagement in the lecture without becoming bored. The students who were not permitted to doodle had a surplus of mentons, causing boredom, and ultimately disengagement from the lecture (Davies & Fortney, 2012).

2. Method:

I built the Lake cognitive architecture to help actualize the menton theory. I built Lake to be composed of three classes, the production system, the menton pools and the parser. Each of these classes are theorized to exist in an individual's mind. Below the three classes, their purpose and how they are connected to the greater system are outlined and explained.

Lake is the architecture, and an individual "model" is designed by the modeler to simulate mental action for a particular task.

2.1 The Production System

Production systems are built to manage the execution of productions. Productions will have conditions required before they are able to execute ("if"), and actions that will happen on

execution (“then”). The production system is responsible for holding productions, calling an evaluation function, which determines which productions’ conditions are met, and returning all the executable productions. The way a production system executes these productions, and the evaluation function used to determine if a production is executable, is unique to the system.

The production system used by Lake was inherited from Hajali, Mekik, & Noronha (2014). They built a production system that would be easily modifiable for most applications. There were very few changes made to this system, since the rest of Lake could be coded in a way that would be compatible with this existing system. However, this production system did have a few known bugs, some of which I patched, and some still exist in the current version of the production system. These bugs will be discussed in more detail in the discussion section of this paper.

The goal of the production system is to execute the productions or gactions (discussed in section 2.1.1) that it is given by the model. Upon creation each production must be bound to an instance of a `productionSystem` (using `production.bind` class method). Each production has a precondition and an action. The precondition is a set of the conditions that must be true for a production to fire. Each cycle, the `productionSystem` will send a list of all the productions to the `Interpreter,` which checks what knowledge currently exists in the `mind`, and returns a dictionary (key, value pairs), where the keys are the productions, and the values are Boolean values describe if the production’s precondition is satisfied and the production is able to fire. This dictionary is passed to `AllSatisfied`, which fires all productions with a true key. When a production fires, its `action` is added to the `mind’s` knowledge. It is important to note, at this point there is not conflict resolution, if the production’s precondition matches the existing

knowledge and the resources it requires are available, then it will fire. If there are multiple productions that are satisfied at the same time, all those productions will fire.

The majority of the production system works without much customization, so long as everything is initialized properly. Since the production system stores the information in a semantic network, it is crucial that all elements are consistent and defined before being organized in knowledge structures (Hajali, Z., Mekik, C., & Noronha, J., 2014).

When setting up a model, there is a specific order in which the components (`mind`, `productionSystem`, `productions`) need to be added to create a successful model. This is because the various components of the model have dependencies within Lake, and it is important to build the model in a way that will allow for all the dependencies to be satisfied. The very first thing that needs to be built is the `mind`, which represents the mind of a given individual. It is simple to build the mind, as all it requires at initialization is a string for a name (e.g. "Mackenzie's Mind"). Once the mind is initialized, the `productionSystem` can be created. The production system must be bound to a mind, which can be done with mind.bind class method, which in addition to the production system takes the menton pool variables (`pools`, `capacities`, `priorities`, `rates`, and `cost`) as additional parameters. The production system also requires initial system knowledge, chunks or goal/action. To see how this looks structurally see Figure 1, where all components are dependent on those connected beneath them, this is represented in code in Figure 2, where `drive`, `lookout`, `observe` and `stop`, are gactions (aka productions, gactions are explained at a later point in this paper).

Figure 1:

Component Dependencies in Lake

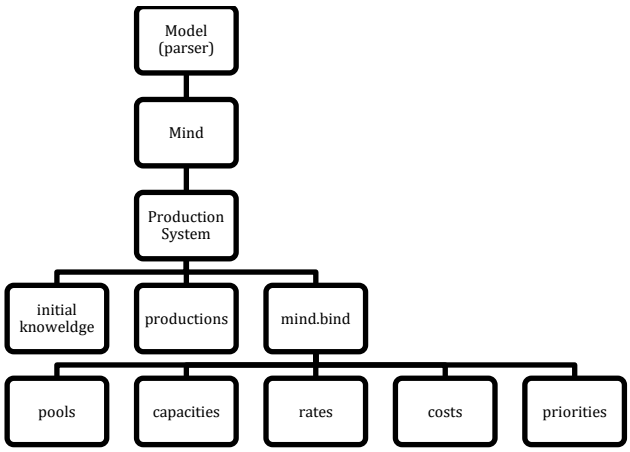

Figure 2:

Initialization of Production System and Dependencies

```
productions = {drive, lookout, observe, stop}
knowledge = Knowledge({Node("No initial system knowledge")})

mind = lake.Mind("test_mind")

pools = {'Perceptual' : 3, 'Motor/Planning' : 3}
capacities = {'Perceptual' : 5, 'Motor/Planning' : 5}
rates = {'Perceptual' : 2, 'Motor/Planning' : 2}
costs = {drive : {'Perceptual' : 1, 'Motor/Planning' : 1},
         lookout : {'Perceptual' : 1},
         observe : {'Perceptual' : 1},
         stop : {'Motor/Planning' : 5}}
priorities = {drive : 1,
              lookout : 1,
              observe : 1,
              stop : 1}

parameters = (pools, capacities, rates, costs, priorities)
system = ProductionSystem(knowledge, productions, mind.bind, parameters)
```

2.1.1 Knowledge Representation

It is very common in cognitive architectures and cognitive modeling to represent goals and actions as separate entities, and to consider goal oriented behaviour as a cornerstone of

rationality (Müller, 2016). Lake, however, does not represent knowledge in this way. Goals and actions are both referred to as "gactions", since goals and actions are commonly interchangeable (Davies, unpublished manuscript). For example, an individual's goal one morning might be to walk the dog. But walking the dog has several sub-goals, such as getting the leash, getting the dog to sit, opening the clasp on the leach, bending over to attach the leash, etc. It is very difficult to identify which of these steps are goals, and which are actions, and ultimately whether something is a goal or an action depends largely on perspective. Because of this, I built Lake to have both goals and actions represented by one structure: a gaction. Whether or not any particular gaction is a goal or an action depends entirely on its relation to the gactions surrounding it and the perspective with which the gaction is being evaluated (Davies, unpublished manuscript). Figure 3 demonstrates how a gaction is represented within a model.

Figure 3:

Gaction Representation in Lake

```
{
  "name": "gaction_name",
  "precondition":[
    {"node": "must"},
    {"node": "exist"},
    {"node": "for me to run"}
  ],
  "action":[
    {"start":[
      {"node": "this starts happening"}
    ]},
    {"stop":[{"node": "this stops happening"}
  ]},
  "priority": 1,
        "req_costs":[
                {"min":[
                        {"visuo-spatial": 200},
                        {"motor/planning": 200}
                ]},
                {"max":[
                        {"visuo-spatial": 600},
                        {"motor/planning": 600}
                ]},
                {"ideal":[
                        {"visuo-spatial": 400},
                        {"motor/planning": 400}
                ]}
        ]

}
```

2.2 Menton Pools

I built the `MentonPool` class to manage the consumption and refreshing of mentons to various menton pools. There can be multiple menton pools (e.g. motor/planning, perceptual, etc.), so long as each pool is bound to a `mind`, and has a unique name. Each pool requires the `current_mentons`, the `replenish_rate` and `capacity`. `Current_mentons` represents the how many mentons are currently in the pool (at initialization). The `replenish_rate` is the constant rate at which mentons refresh per unit time. `Capacity` refers to a menton pool's `capacity`, the maximum number of mentons that can be contained in the specific pool.

The `replenish_rate`, as well as the `capacity`, are arbitrary or best guess numbers in the current models, and may be changed by the user. Empirical research needs to be conducted to be able to create a range on what these numbers should be.

The goal is to create an architecture where models can be easily built and updated, and once there is research supporting what these ranges should be, the models can be updated accordingly. Hajali, Mekik and Noronha (2014) made a first attempt at defining the range for menton `replenish_rate`. They found an average of 7.925 mentons per second during a go/no-go trial. Their work also suggests that more challenging tasks lead to a higher `replenish_rate` (e.g. 1.35 mentons/second when listening to the radio and driving vs. 4.75 mentons/second when talking on the phone while driving). This is something that should be explored with further testing, and added to a future iteration of Lake.

Per the menton theory, there are several things that would influence a gaction to fire, such as the state of the system, finding a matching memory, goal recruiting a gaction, a gaction matching or satisfying a goal, strength of a gaction and the activation level of a gaction (Davies

& Fortney, 2012). For Lake to properly represent the theory, all of these things must be present. In the version of Lake I have built thus far, I have implemented some, but not all of these triggers. Implementing the remainder of these triggers is an important next step for Lake. I implemented Lake to trigger a gaction if the gaction's requirements can be matched to existing memories within the agent. For example, take the two heavily simplified gactions defined in Figure 4. Assuming the goal of the system is to walk and chew gum, and both the gactions' preconditions were met at the same time, both gactions would be triggered to fire. If there were enough mentons in the visuo-spatial pool (minimum: 450) and there were enough mentons in the motor/planning pool (minimum: 500), then both gactions would fire without issue. I also built in Lake the functionality to prevent firing of gactions based on their menton requirements and their priority level. To continue from the previous example, if the visuo-spatial pool only had 425 mentons, it would not be able to fire both of the gactions. I built Lake to handle conflicts such as this with its prioritize method in the `Mind` class. The `prioritize` class takes a list of all the gactions that are currently matched to be fired, and ranks them based on their priority. That sorted list is then passed to the executable and deduct classes, which will execute the gactions where there are enough mentons to fire. Following this example, the gactions would be ranked "walk" over "chew (gum)"; "walk" would fire, leaving only 25 mentons in the visuo-spatial pool. This is not enough mentons to fire the next gaction: "chew (gum)", so the process finishes, the pools update, and the model continues.

Figure 4:

Example Gactions

```
{
        "name" : "chew (gum)",
        "precondition":[
                {"node": "chewing gun in mouth"}
        ],
        "action":[
                {"start":[
                        {"node": "chewing gum"}
                ]},
                {"stop":[]}
        ],
        "priority":1,
        "req_costs":[
                {"min":[
                        {"visuo-spatial": 50},
                        {"motor/planning": 100}
                ]},
                {"max":[
                        {"visuo-spatial": 200},
                        {"motor/planning": 400}
                ]},
                {"ideal":[
                        {"visuo-spatial": 125},
                        {"motor/planning": 250}
                ]}
        ]
},
{
        "name" : "walk",
        "precondition":[
                {"node": "desire (walk)"}
        ],
        "action":[
                {"start":[
                        {"node": "walk"}
                ]},
                {"stop":[]}
        ],
        "priority":3,
        "req_costs":[
                {"min":[
                        {"visuo-spatial": 200},
                        {"motor/planning": 200}
                ]},
                {"max":[
                        {"visuo-spatial": 600},
                        {"motor/planning": 600}
                ]},
                {"ideal":[
                        {"visuo-spatial": 400},
                        {"motor/planning": 400}
                ]}
        ]
}
```

There is a great deal of room for improvement on this aspect of the system. These improvements will be outlined, along with some direction on how these improvements could be implemented in the discussion section of this paper.

2.3 The parser class

I built the parser class as a middleware that allows individuals who are less familiar with Python or object-oriented development the ability to leverage the architecture by creating unique models. One of the goals of creating an architecture for the menton theory, is to be able to create and test models in the architecture against psychological studies. Psychologists running these studies do not always have a background in software development, and therefore the parser class was created so the models can be built quickly and easily, regardless of the individual's software development skills.

This parser allows the user to input a JSON file as opposed to writing a Python file. JSON files are designed to be easy to read, learn and build regardless of computer science background, which is why I chose it as the format for Lake's model development. There are two examples, the gaction Figure 5a is JSON while the gaction in Figure 5b is the raw Python code. I laid the JSON code in a way that follows the model's creator's thought process when designing a gaction. Although the JSON code still requires some ramp-up to understand how to build a model, there are no requirements to understand `imports`, `loops` or `function calls` and `parameters`, which is a requirement for writing a model directly within python. Although further simplifying model creation remains a goal for future iterations of Lake, I chose to start with building a parser for JSON because syntax errors will be easy to locate when writing the

model in any simple and free text editor (such as Sublime Text 3), as the location of the error will be highlighted.

Figure 5a: *Python Gaction*

```
stop = Production(
    Precondition(
        {Node("Light flash"),
         Node("Observed light"),
         Arrow((Node("Is"), Node("Light flash"), Node("Red")))},
        {Node('Stopped')}),
    Action({Node("Stopped")}))

productions = {drive, lookout, observe, stop}
knowledge = Knowledge({Node("No initial system knowledge")})

engine = lake.Mind("test_mind")

pools = {'Perceptual' : 3, 'Motor/Planning' : 3}
capacities = {'Perceptual' : 5, 'Motor/Planning' : 5}
rates = {'Perceptual' : 2, 'Motor/Planning' : 2}
costs = {drive : {'Perceptual' : 1, 'Motor/Planning' : 1},
         lookout : {'Perceptual' : 1},
         observe : {'Perceptual' : 1},
         stop : {'Motor/Planning' : 5}}
priorities = {drive : 1,
              lookout : 1,
              observe : 1,
              stop : 1}

parameters = (pools, capacities, rates, costs, priorities)
system = ProductionSystem(knowledge, productions, engine.bind,
parameters)


system.step()
print "State of the Mentons in step 1: "
for i in range(len(engine.mentonPools)):
    print(str(engine.mentonPools[i].name) + ": " +
str(engine.mentonPools[i].mentons))
system.knowledge.add({Node('Light flash'),
                      Arrow((Node("Is"),
                             Node("Light flash"),
                             Node("Red")))})
for i in range(3):
    system.step()
    print "State of the Mentons in step " + str(i+2) + ": "
    for i in range(len(engine.mentonPools)):
        print(str(engine.mentonPools[i].name) + ": " +
str(engine.mentonPools[i].mentons))
```

Figure 5b:

JSON Gaction

```
{
        "name" : "walk",
        "precondition":[
                {"node": "desire (walk)"}
        ],
        "action":[
                {"start":[
                        {"node": "walk"}
                ]},
                {"stop":[]}
        ],
        "priority":3,
        "req_costs":[
                {"min":[
                        {"visuo-spatial": 200},
                        {"motor/planning": 200}
                ]},
                {"max":[
                        {"visuo-spatial": 600},
                        {"motor/planning": 600}
                ]},
                {"ideal":[
                        {"visuo-spatial": 400},
                        {"motor/planning": 400}
                ]}
        ]
}
```

3. Discussion:

As discussed, Davies and Fortney (2012) suggest that the mind uses mentons to execute various tasks. This theory relates and discourses with the other theories introduced at the onset of this paper. The menton theory can align with the bottleneck theory (Pashler & Johnston, 1998), if there was one menton pool. The more tasks requiring mentons, the fewer mentons available for execution of tasks, causing a bottleneck. Although the menton theory aligns with bottleneck theory and the central capacity theory (Kahneman, 1973), in there being a single resource that is responsible for executing cognitive tasks, when the central capacity theory fails to account for the results of dual interference (Sanders, 1997), the menton theory holds. The menton theory is able to account for the results of dual interference tasks despite it suggesting a

single resource (mentons), as it does not suggest there being a single menton pool, rather there could be multiple pools for various aspects of the mind. The concept of there being multiple pools of mentons, closely aligns with the multiple resources theory, which Wickens (2002) developed to address the results of dual task interference when existing theories failed. Multiple resource theory's suggestion that dual task interference is a result of the same *types* of tasks causing interference, such as verbal and verbal tasks causing more interference than verbal and visual tasks, aligns well with the menton theory as the more tasks drawing from the same menton pool, results in less mentons available for each task.

This current architecture achieves the initial steps towards implementing the menton theory as a cognitive architecture, although there are still many steps that need to be taken towards fully implementing the theory. The primary focus going forward with this architecture would be to implement quality degradation when the menton pool is being fully taxed. For example, if you have the 3 gactions, as in Figure 6, trying to execute at the same time, you should see the quality of the tasks degrade relative to their priority, allowing for all three to fire simultaneously. This would align more closely with how one interacts with the world. If one was walking down the street, chewing gum, when their phone rang they wouldn't necessarily need to stop walking or chewing gum to be able to start talking on the phone, rather they might slow their walking or gum chewing when answering their phone. The current version of Lake does not support this. The current architecture always uses the "ideal" number of mentons required by a gaction, when selecting which gactions to fire. I did take this into consideration during the development of Lake, and provided place holder values for the minimum and maximum required mentons for the gaction's execution. This will allow a future developer to write just one function to add this functionality, and not have to change any object structures.

Figure 6:
Gaction Example

```
{
        "name" : "chew (gum)",
        "precondition":[
                {"node": "chewing gun in mouth"}
        ],
        "action":[
                {"start":[
                        {"node": "chewing gum"}]},
                {"stop":[]}
        ],
        "priority":1,
        "req_costs":[
                {"min":[
                        {"visuo-spatial": 50},
                {"motor/planning": 100}]},
                {"max":[
                        {"visuo-spatial": 200},
                {"motor/planning": 400}]},
                {"ideal":[
                        {"visuo-spatial": 125},
                {"motor/planning": 250}]}]
},
{
        "name" : "walk",
        "precondition":[
                {"node": "desire (walk)"}
        ],
        "action":[
                {"start":[
                        {"node": "walk"}]},
                {"stop":[]}
        ],
        "priority":3,
        "req_costs":[
                {"min":[
                        {"visuo-spatial": 200},
                {"motor/planning": 200}]},
                {"max":[
                        {"visuo-spatial": 600},
                {"motor/planning": 600}]},
                {"ideal":[
                        {"visuo-spatial": 400},
                {"motor/planning": 400}]}]
},
{
        "name" : "talk on phone",
        "precondition":[
                {"node": "phone ringing"}
        ],
        "action":[
                {"start":[
                        {"node": "talk on phone"}]},
                {"stop":["phone ringing"]}
        ],
        "priority":2,
        "req_costs":[
                {"min":[
                        {"visuo-spatial": 50},
                {"motor/planning": 100}]},
                {"max":[
                        {"visuo-spatial": 100},
                {"motor/planning": 600}]},
                {"ideal":[
                        {"visuo-spatial": 75},
                {"motor/planning": 350}
                ]]
```

Davies and Fortney's (2012) theory suggests that in situations where there are more mentons required than the pool has available, the quality of some tasks might decrease, until the mentons are once again in the black. To represent this in the architecture, it is important to consider not only how it should work in the code, but how it would work in reality. There are two primary situations that should be considered when building this functionality. The first is when there are already gactions running that are competing with a new gaction that will then require above the maximum available mentons. For example, washing dishes and listening intently to the radio are executing, when some other agent enters the room to start a conversation. This conversation will require more mentons than are remaining in the pool. A function needs to be built that will assess the priority of the gactions that want to fire, and determine if one or more of the gactions currently firing (dishes and radio) need to lend some of its mentons to the new gaction (conversation). It then needs to determine (based on priority, and eventually desire, environment and any number of other factors) how many mentons each gaction will receive. It should then gradually move these mentons around to satisfy the new allocations. The rate at which these mentons are moved is not currently known. Research should be done in this area to determine at what frequency mentons can move between gactions. The second situation to consider, is if there are no gactions currently firing, and the total gactions about to fire require more than the total number of mentons. This situation is much easier to handle, as it only requires the system to determine how many mentons to allocate to each task. Development of a formula is required to determine what percentage of mentons each gaction gets based on some series of variable (e.g. priority, min/max/ideal mentons required etc.).

Another item which is important to implement in future iterations of this architecture is partial matching. Some examples of partial matching include: peoples' faces change over time,

but are still recognizable, a book on a shelf is the same as a book on the floor, two times three is the same as three times two (Anderson & Lebiere, 1998). These are all examples of partial matching, which are not currently covered by lake. As of right now, Lake fires gactions based on their precondition matching what chunks and gactions currently exist using a semantic network in the `productionSystem`. This semantic network requires perfect matching. For example, "2 fish" will not match with "two fish" and "yellowish book" will not match with "bright yellow book" or "yellow book". Capabilities for partial matching should be a high priority in future iterations of Lake.

Beyond the theory related functionality that needs to be added to improve this architecture, there exist a few technical bugs that will likely be minimized or eliminated as future iterations of this system are developed. The primary quirk with is involved with the `productionSystem`. There are various types of productions that can be created, chunk-1s which are when there is a single piece of information in the production (e.g. "read book"), a chunk-3 is when there is a join between two pieces of information (e.g. "light is green") where *is* acts as an arrow connecting light and green. The problem arises within the `productionSystem` when you want to search for a node, but the node only appears in the arrow section of a chunk-3, it will not be found. This problem should be resolved by either changing the way chunk-3s are stored, or by adding addition search functionality to the `productionSystem`'s `find` class, to look beyond the surface nodes.

Another technical quirk in the production system, is that it does not currently allow for the use of the "or" operator in a gaction's precondition. For example, if there was some gaction "sip coffee" that should run when the agent is either "thirsty" or "tired" or "cold". This is not currently supported, and can only be represented by either three separate "sip coffee" gaction

(each with a different precondition) or by having one “sip coffee” gaction that requires all three of these preconditions to fire.

4. Conclusion and future work:

This paper, the associated models and architecture, represent the initial steps towards bringing the menton theory of mental resources to an application reality. For the completion of my thesis, I built the Lake cognitive architecture, apart from the production system. This includes, the menton engine, the architecture of the mind and tweaked minor pieces of the production system to allow all the components to work together. There are a several quirks with this architecture, many of which will be eliminated as additional functionalities are added to fully represent the theory. As discussed above, this application does not currently account for theory requirements for task quality degradation and this should be one of the main focuses of future work.

References:

Anderson, J., & Lebiere, C. (1998). *The atomic components of thought.* Mahwah, NJ: Lawrence Erlbaum Associates.

Basil, M. (2012). *Multiple resource theory.* Encyclopedia of the Sciences of Learning. 222-223. Doi: 10.1007/978-1-4419-1428-6

Davies, J. & Fortney, M. (2012). The menton theory of engagement and boredom. Poster Collection: The First Annual Conference on Advances in Cognitive Systems, pp. 131–143.

Hajali, Z., Mekik, C., & Noronha, J. (2014). To Quantify Mentons, Model Dual Task Interference. Unpublished Manuscript.

Heuer, H. (1985). Some points of contact between models of central capacity and factor-analytic models. *Acta Psychologica, 60,* l35-155

Laird, J., & MIT CogNet. (2012). *The soar cognitive architecture*. Cambridge, Massachusetts: The MIT Press.

Pashler, H., & Johnston, J. C. (1998). Attentional limitations in dual task performance. In H. Pashler (Ed.), *Attention*, (pp. 155-189). Hove, U.K.: Psychology Press

Müller, V. C. (Ed.). (2016). Risks of Artificial Intelligence. London, GB: Chapman and Hall/CRC. Chapter 7. Retrieved from http://www.ebrary.com.proxy.library.carleton.ca

Navon, D., & Gopher, D. (1979). On the economy of the human information processing system. *Psychological Review, 86,* 214—255

Stewart, T. C., & West, R. L. (2007). Deconstructing and reconstructing ACT-R: Exploring the architectural space. *Cognitive Systems Research, 8*(3), 227-236. doi:10.1016/j.cogsys.2007.06.006

Tombu, M., & Jolicœur, P. (2003). A central capacity sharing model of dual-task performance. *Journal of Experimental Psychology: Human Perception and Performance, 29*(1), 3-18. doi:http://dx.doi.org.proxy.library.carleton.ca/10.1037/0096-1523.29.1.3

Wickens, C. D., Multiple Resources and Mental Workload. Volume: 50 issue: 3, page(s): 449-455. Article first published online: June 1, 2008;Issue published: June 1, 2008 DOI: https://doi-org.proxy.library.carleton.ca/10.1518/001872008X288394

Wickens (2002). Multiple resources and performance prediction. *Theoretical Issues in Ergonomic Sciences, 3*, 15-177.