Cognitive Architecture in Mobile Music Interactions

Nate Derbinsky
Computer Science & Engineering Division
University of Michigan
2260 Hayward Ave
Ann Arbor, MI 48109-2121
nlderbin@umich.edu

Georg Essl
Electrical Engineering & Computer Science and Music
University of Michigan
2260 Hayward Ave
Ann Arbor, MI 48109-2121
gessl@eecs.umich.edu

ABSTRACT

This paper explores how a general cognitive architecture can pragmatically facilitate the development and exploration of interactive music interfaces on a mobile platform. To this end we integrated the Soar cognitive architecture into the mobile music meta-environment urMus. We develop and demonstrate four artificial agents which use diverse learning mechanisms within two mobile music interfaces. We also include details of the computational performance of these agents, evincing that the architecture can support real-time interactivity on modern commodity hardware.

Keywords

mobile music, machine learning, cognitive architecture

1. INTRODUCTION

How can contemporary work in machine learning and cognitive architectures be used in mobile music interactions? Here we integrate a contemporary cognitive architecture with an emerging mobile music environment and show the pragmatic use of various learning stategies in this context.

The introduction of interactive music-making techniques has shown some impressive outcomes. Fiebrink *et al* [6] have demonstrated that supervised machine learning can be used to define interactive gesture-based music applications on laptops. However the introduction of comparable ideas to mobile music interaction is lacking. Current mobile smart devices are different from laptops in the kinds of interactions that are natural to perform on them and the kinds of sensors that are available on them. For example hand gestures are a rather natural mode of engagement with a mobile device, whereas accelerometer-based interactions on laptops are possible but have a distinctly different flavor. Further mobile smart devices are available to a larger demographic than laptops suggesting the need to support them as primary computational platforms [4].

In this paper, we explore interactive learning and musical expression on mobile devices. In contrast to prior work that applied specialized machine learning algorithms, we use a cognitive architecture, a system that efficiently and generally integrates multiple learning and memory modules for use across numerous tasks.

NIME'11, 30 May–1 June 2011, Oslo, Norway.

2. COGNITIVE ARCHITECTURE

The central goal of artificial intelligence is the development and understanding of intelligent agents, autonomous entities that observe and act within an environment, applying human-level reasoning capabilities to achieve their goals. Many researchers in the field, however, do not work directly at the level of generally intelligent agents, but instead strive to understand one or more sub-problems in specific contexts, such as machine learning, the study and development of algorithms to find patterns in empirical data; planning and reasoning, especially under uncertainty; and computational processing and generation of natural language.

By contrast, research into cognitive architecture aims to develop and understand human-level intelligence across a diverse set of tasks and domains [8]. A cognitive architecture is a specification of those aspects of cognition that remain constant throughout the lifetime of an agent. These fixed components include short- and long-term memories of the agent's beliefs, goals, and experience; the representation of elements contained within these knowledge stores; functional processes that apply agent knowledge to produce behavior; and learning mechanisms that adapt agent knowledge over time. Cognitive architecture applies a systems-level approach to artificial intelligence research, investigating how the integration of numerous computational mechanisms supports complex and adaptive behavior.

Diverse cognitive architectures have been developed over the last forty years, but nearly all specific cognitive architecture research efforts strive towards at least one of the following three goals: (1) biological plausibility, (2) psychological plausibility, and (3) agent functionality. For instance, systems such as Leabra [10] attempt to computationally explore how intelligence arises from circuits of neurons and how architectural mechanisms and processes correspond to neurobiological data regarding brain regions and topological connectivity. By contrast, systems such as EPIC [9] and ACT-R [1] are typically applied at a layer above biological mechanisms and attempt to capture and model details of human performance, such as behavioral timing and memory recall errors, in a wide range of cognitive tasks. Finally, architectures like Soar [7] strive to understand how human-level intelligence arises from computational architecture and are typically applied as an effective path to building broadly capable artificial agents.

There are two primary appeals of considering cognitive architectures for music performance. The first involves quality of interaction with a learning system. The response of an interaction may involve familiar characteristics, such as remembering or forgetting musical phrases. Agent functionality can offer the appearance of such cognitive function in a way that a performer can potentially relate to, hence making the machine learning process itself more intuitive. The second reason is one of development pragmatism. Typical

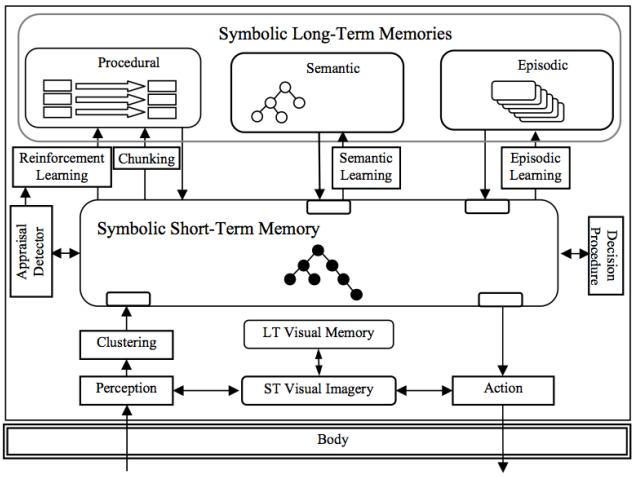


Figure 1: The structure of the Soar cognitive architecture.

applications of machine learning techniques involve specialized algorithms and thus there is significant burden on developers to implement one or more algorithms, tune them for a particular task, including any integration issues that arise, and finally employ optimization techniques such that the algorithms scale to complex problems, an especially difficult challenge on mobile platforms. These burdens are lessened with the application of cognitive architecture wherein the locus of development is declaring agent knowledge and goals.

In addition to research goals, individual cognitive architectures deviate along numerous dimensions. We considered two metrics in selection of the candidate architecture. First, to explore complex musical expression, we sought an architecture that could process over, as well as reason and learn about, diverse information sources, including temporal sequences of musical notes and declarative rules of music composition. Second, to support interactive mobile tools, we we sought an architecture that could bring these knowledge sources to bear while maintaining real-time decision making. Thus, we applied and evaluated Soar [7], a functionally driven cognitive architecture.

3. SOAR

Soar is a cognitive architecture that has been used extensively for developing artificial intelligence applications and modeling human cognition. One of Soar's main strengths has been its ability to efficiently represent and bring to bear large bodies of symbolic knowledge to solve diverse problems using a variety of methods [7]. Soar supports a variety of programming languages (such as C++, Java, and Python) on all major operating systems (including Windows, Mac OS, Linux, and iOS) and has been interfaced in diverse execution environments, including game systems and robotics simulation and hardware platforms.

Figure 1 shows the structure of Soar. At the center is a symbolic working memory, represented as a graph, that captures the agent's current state. Perception from the world, such as sights, sounds, or contact, delivers symbolic structures to working memory. The long-term memories retrieve information based on the contents of this working memory and add, delete, or modify these structures. The procedural memory, encoded as if-then rules, captures the agent's knowledge of when and how to perform actions, both internal, such as deliberately querying other long-term memories, and external, such as the production of sound through speakers or control of robotic actuators. This knowledge can be tuned over time by the integrated reinforcement learning [11] mechanism, which adjusts the selection of actions in an attempt to maximize receipt of reward. The semantic long-term memory encodes general facts about the world, which may be pre-loaded from existing knowledge bases, while episodic memory incrementally builds an autobiographical history of agent experience. As evident in Figure 1, Soar has additional memories and processing modules; however, they are not evaluated in this paper, and are not discussed further.

Processing in Soar is decomposed into a sequence of decisions. The basic *decision cycle* is to process input, fire rules that match, make a decision, fire rules that apply the decision, and then process output commands and retrievals from long-term memory. The time to execute this processing cycle determines reactivity and so our evaluation will include (1) the number of decisions that were made to complete the task, (2) the average amount of time to execute each cycle, (3) the maximum amount of time required for any cycle, and (4) the total CPU time consumed by the Soar agent completing the task.

4. INTEGRATING SOAR IN URMUS

UrMus [5] is set up to provide a flexible system to receive input and organize visual and other content in response. The main organizing element for multi-touch input as well as visual output are regions. They allow maximum flexibility in designing interactions. Hence it felt natural to associate an instance of Soar with regions. Each region can have an instance of Soar attached, hence one can have one or more agents for each input element and agents for each movable visual element. Other media elements can be realized by having region-based events instantiate Flowboxes (elements of UrMus's dataflow engine). This means it is easy to have objects that independently navigate, say, the screen-space to have independent cognitive models running.

The Soar kernel is implemented in C++ and we have integrated the architecture with urMus via a minimal Lua interface that allows urMus to supply perception to the agent, including touch events from the user, read actions decided upon by the agent, such as producing a note, and execute arbitrary commands, such as to control the agent's run state and illicit debugging and computational performance information.

To see how this works, let us consider the following simple example code. As said, each region can have a Soar agent attached to it. In order to do anything meaningful, this agent will need a rule set to be loaded:

```
r = Region()
r:SoarLoadRules("simon-rl", "soar")
```

In order to learn, a percept (in form of a symbolic constant) is created in Soar's input data structure and a learning step is executed. In this case a notion of time is learned that is derived from a user interaction. The input is deleted when done:

```
timeWme = r:SoarCreateConstant(0,
                    "time", clickcount)
r:SoarExec("step "..delayDecisions)
r:SoarDelete(timeWme)
```

In order to generate output after learning, one can read the output from Soar's output data structure:

```
taskWme = r:SoarCreateConstant(0,
                    "task", "generate")
name, params =  backdrop:SoarGetOutput()
```

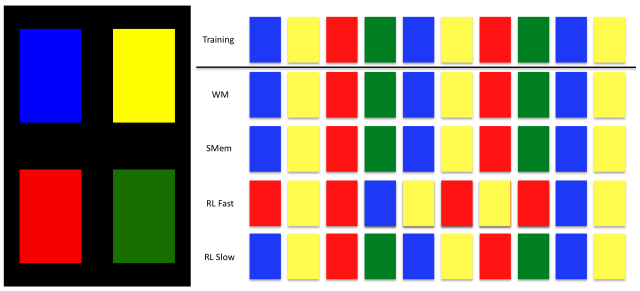


Figure 2: Simon demonstration with example input/output sequences.

```
result = params.output
r:SoarSetOutputStatus(1)
r:SoarDelete(taskWme)
```

Here `output` is the entry of interest. There are also other functions that help control a Soar agent, such as halting the agent with the `r:SoarFinish()` function and reinitializing the agent anew with the `r:SoarInit()` command.

5. DEMONSTRATIONS

We now present two musical demonstrations implemented in the UrMus environment [5] using Soar 9 [7] and deployed to the iOS platform on iPhone, iPad, and iPod Touch hardware. These tools are not intended to represent state-of-the-art music interface design, but instead demonstrate how cognitive architecture can facilitate the development of interactive, novel musical tools on mobile platforms.

5.1 Simon

Our first demonstration was to implement Soar agents playing Simon, a game of memory skill [2] illustrated in Figure 2. In this demonstration, the user inputs a sequence, selecting from four colored buttons, each of which emits a different musical tone. The button presses are provided to the Soar agent sequentially in real-time and after input is complete, the Soar agent generates an arbitrary length of musical response, based upon its knowledge and learning of the input stream. The focus of this demonstration is to explore how utilizing different memory models, both short- and long-term, can affect development of musical interfaces. For clarity, the Soar cognitive architecture is held constant for all demonstration agents below, whereas the agent's initial procedural knowledge, encoded as if-then rules, is what is altered such as to distinguish each agent's behavior.

In this task, our first evaluation metric was accuracy of the agent's response - to what extent does the agent reproduce the input sequence? While a challenge for humans, especially over long input sequences, one could imagine basic algorithms that could store and generate a fixed length button string. Thus, our second, more interesting evaluation dealt with output generativity, or the degree to which the agent could produce novel output, while adhering to the "spirit" of the input sequence.

The first agent we developed appended new button inputs to an endless linked list within the agent's working memory. As expected, the result of this agent was perfect input reproduction, at the cost of learning no generalization over the input. After receiving input and producing an output sequence of 10 buttons, the agent required 415 decision cycles, averaging 0.053 milliseconds per decision with a maximum of 1 millisecond for a decision, for a total of 0.022 CPU seconds.

The next agent applied an instance-based learning approach, storing all inputs to its semantic memory. We encoded each note as a simple (previous note, next note) pair and thus generation of new musical sequences simply retrieved the "next" note conditioned upon the last note played. Figure 3 captures the state of the agent's semantic memory after it had heard the sequence of three notes: A, B, C (where one/two refer to previous/next and *nil* refers to the beginning of the sequence). For very short, distinct button sequences (fewer than four buttons, no repetition), it is possible for the agent to perfectly reproduce the input. However, the common case is for ambiguities to arise in the input sequence, which are disambiguated via a recency bias in memory retrieval [3]. This agent required 278 decisions at an average of 0.086 milliseconds per decision, requiring a maximum of 1 millisecond in a decision, for a total of 0.024 seconds of CPU time.

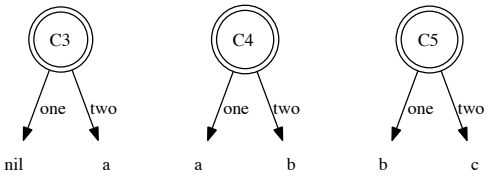


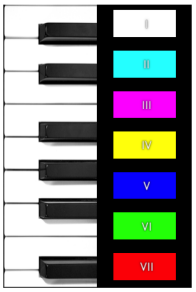
Figure 3: Simon semantic memory representation.

Our final agent applied Soar's reinforcement learning mechanism to this memory task. During the user's input, the agent "practiced" the known sequence, given the same state representation as the semantic memory agent, self-rewarding for reproducing the correct input. For instance, the following rule captures some practiced knowledge after the agent heard the sequence of three notes: A, B, C.

```
If
   no previous note was heard AND
   the agent is considering producing note C
Then
   the expected value of this decision is -5
```

The value at the conclusion of the rule was updated over time based upon experience and practice. The amount of practice was directly proportional to the amount of time between user inputs. As a result, the generated musical output was affected by the sequence of buttons the user pressed, the time taken to input the sequence, and the probabilistic application of the agent's learned music generation policy. We found that given enough time between button presses (about 1 second), the agent would frequently reproduce most of the initial input sequence, while occasionally deviating to produce novel, probabilistically derived subsequences. We provided the same input sequence to the agent twice, varying only the amount of time between button presses (such as to change available practice time). The agent given little practice ran for 827 decisions, averaging 0.145 milliseconds per decision with a maximum of 1 millisecond for a decision, totally 0.120 seconds of CPU time. The agent with more time ran for 1133 decisions, averaging 0.176 milliseconds per decision with a maximum of 1 millisecond, totalling 0.199 seconds of CPU time.

The agent performances in the Simon demonstration are summarized in Figure 2 for a particular input sequence ("Training). The working memory ("WM") agent perfectly reproduces the input, as it cannot generalize. The semantic memory agent ("SMem"), with a limited instance-based representation, primarily reproduces the input (as seen in Figure 2), with small deviations when the next note is not uniquely determined by the previous note. The reinforcement learning agent is shown with two amounts of practice



Training	
Chord	Melody
I	EGGE
V	DGGD
VI	CEEC
III	BEEB
IV	ACCA
I	GCCG
IV	ACCE
V	DCDD

Output 1	
Chord	Melody
I	GGGC
V	CGGG
VI	CCEC
V	DCDD
VI	ECEC

Output 2	
Chord	Melody
I	GGGC
V	CGGG
VI	CCEC
V	DCDD
VI	ECEC

Figure 4: The interface of the music generation urMus/Soar implementation (left). Training and two generated results using the reinforcement learning.

time ("RL Fast" and "RL Slow"), illustrating probabilistic differences from the training sequence. If the learning time is fast the sequence is more likely to deviate from transitions seen in learning, whereas longer learning will reinforce transitions that are seen frequently hence lead to sequences that more closely resemble the original. However, these models are probabilistic and hence do not guarantee reproduction. For musical purposes this is interesting because it means that learning the rules of production are reinforced but variation is retained.

5.2 Mobile Music Generation and Interaction

Our next demonstration tasked the agent with generating simple musical scores after perceiving a sequence of chords accompanying musical notes (see interface in Figure 4). To simplify the quantization problem, the time scale of the input was fixed (one chord per 4 notes). Once again, our evaluation considered both the accuracy of music reproduction, as well as novelty of musical generation.

For this demonstration, we extended our reinforcement learning agent from the Simon task such as to simultaneously learn chord sequencing, note sequencing, and note-chord association. The following are two representative rules learned by the agent:

```
If
   the current chord is C-E-G AND
   the previous note played was G AND
   the agent is considering producing note C
Then
   the expected value of this action is -8

If
   the previous chord was C-E-G AND
   the agent is considering chord E-G-B
Then
   the expected value of this decision is -8
```

The agent's practice time (limited by real-time interaction with the user) was split between learning chord sequencing, note sequencing, and note-chord association. We found that given sufficient "practice" time (about 100 decisions between notes), the Soar agent was able to associate notes with chords and produce similar chord sequences as the input, though note sequencing was unimpressive in reproduction, nor generative quality. In this task, our agent required 360 decisions, averaging 0.217 milliseconds per decision and requiring a maximum of 1 millisecond for a decision and a total of .118 seconds of CPU time.

An example training set of eight chords with four note monophonic melodies each and two generated outputs of five chords with four note melodies each are shown in Figure 4. Each run of Soar will generate a new output and note that the adherence to learned rules is not yet very strict. Notes that do not strictly belong to the underlying chord are played. The training set contains one such exception. With repeated training the melodies become more reflective of the input. This model can be run offline or interactively in a call and response scheme. The user plays a chord and four notes and the system will generate the same based on what it has learned so far. Over time the call and response duet locks into a more stable style as the learning algorithm reinforces the observed rules of the played call melodies.

6. CONCLUSIONS

In this paper we showed how the integration of a cognitive architecture and a mobile music platform can lead to novel forms of interactive music expression. We developed two systems that demonstrate how interactive musical tools can benefit from complex, integrated applications of machine learning algorithms (such as reinforcement learning) and instance-based learning (such as declarative retrievals from semantic memory). We also showed that the Soar cognitive architecture is sufficiently efficient to support interactive musical tools on a mobile platform. These demonstrations, however, do not begin to explore the space of possibilities a cognitive architecture offers to the development of novel mobile musical tools.

7. REFERENCES

[1] J. R. Anderson, D. Bothell, M. D. Byrne, S. Douglass, C. Lebiere, and Y. Qin. An Integrated Theory of the Mind. *Psychological Review*, 111:1036–1060, 2004.

[2] R. H. Baer and H. J. Morrison. Microcomputer Controlled Game - US Patent 4,207,087, 1980.

[3] N. Derbinsky, J. E. Laird, and B. Smith. Towards Efficiently Supporting Large Symbolic Declarative Memories. In *Proceedings of the 10th International Conference on Cognitive Modeling*, 2010.

[4] G. Essl. Mobile Phones as Programming Platforms. *Proceedings of the First International Workshop on Programming Methods for Mobile and Pervasive Systems*, 2010.

[5] G. Essl. UrMus-an environment for mobile instrument design and performance. *Proceedings of the International Computer Music Conference*, 2010.

[6] R. Fiebrink. *Real-time human interaction with supervised learning algorithms for music composition and performance*. Disseration, Princeton University, 2011.

[7] J. E. Laird. Extending the Soar Cognitive Architecture. In *Proceedings of the First Conference on Artificial General Intelligence*, Memphis, TN, 2008. IOS Press.

[8] P. Langley, J. E. Laird, and E. Rogers. Cognitive Architectures: Research Issues and Challenges. *Cognitive Systems Research*, 10(2):141–160, 2009.

[9] D. E. Meyer and D. Kieras. A Computational Theory of Executive Control Processes and Human Multiple-Task Performance. Part 1: Basic Mechanisms. *Psychological Review*, 1997.

[10] R. C. O'Reilly and Y. Munakata. *Computational Explorations in Cognitive Neuroscience: Understanding the Mind by Simulating the Brain*. MIT Press, Cambridge, MA, 2000.

[11] R. S. Sutton and A. J. Barton. *Reinforcement Learning: An Introduction*. 1998.