In Proceedings of ICCM - 2007- Eighth International Conference on Cognitive Modeling. 313 - 318. Oxford, UK: Taylor & Francis/Psychology Press.

Cognitive Redeployment in ACT-R: Salience, Vision, and Memory

Terrence C. Stewart (terry@ccmlab.ca)
Robert L. West (robert_west@carleton.ca)
Carleton Cognitive Modelling Lab
Institute of Cognitive Science, Carleton University
1125 Colonel By Drive, Ottawa, Ontario, K1S 5B6 Canada

Abstract

Cognitive redeployment is the idea that an important part of the evolution of cognition is the adaptation and re-use of existing cognitive modules for new purposes. In this paper, we apply this idea to the ACT-R declarative memory system and the ACT-R visual system. We have developed a common underlying implementation of these systems as part of the Python ACT-R project. As a result, we can apply aspects of vision to memory and vice-versa. In particular, we apply the concepts of base-level learning, spreading activation, and partial matching from declarative memory to vision. We also apply the recent visual saliency model from vision to declarative memory. This results in a variety of new models of existing phenomena which rest on a simpler and more general theoretical framework.

Introduction

In previous work (Stewart & West, 2006; Stewart & West, in press), we have argued that it is useful to deconstruct the ACT-R cognitive architecture (Anderson & Lebiere, 1998) into its more basic components. By combining these components in various ways, we can explore the effects of *architectural* changes, as well as the more commonly explored effects of numerical parameter changes. In other words, instead of just adjusting values such as latency or noise, it is possible to investigate the effects of adding multiple memory systems, multiple production systems, and new buffers for communication. This system is called Python ACT-R and is integrated with a complete set of tools (CCMSuite) for creating experimental environments, running parallel simulations, and performing data analysis. All source code and experimental data can be found at <http://ccmlab.ca/ccmsuite.html> including the models described here.

In this paper, we explore the implications of treating the ACT-R declarative memory system and the ACT-R visual system as being built out of the same underlying components. These systems are traditionally treated as quite separate in ACT-R; indeed, until recently the visual system was part of an add-on component called ACT-R/PM. However, these systems have a similar interface, in that they both can be given requests to find items that fit a particular pattern. The precise algorithms for determining which items are found by the requests do differ between vision and declarative memory, but we show in this paper that there are underling similarities that allow us to combine these algorithms, and apply the declarative memory ones to vision and the vice versa.

The goal of this work is to determine whether there is an advantage to treating vision and memory as systems built from the same type of generic system. By consolidating the models of these two phenomena and exploring their similarities (and differences), we hope to arrive at a more parsimonious cognitive model.

Cognitive Redeployment

If a generic memory system can be shown to accommodate both the vision and the declarative memory systems within the ACT-R cognitive architecture, then it can be seen as an example of *cognitive redeployment*. Michael Anderson (2007) argues that “the brain evolved by preserving, extending, and combining existing network components, rather than by generating complex structures de novo.” His *massive redeployment* hypothesis (Anderson, forthcoming) states that “cognitive evolution proceeds in a way analogous to component reuse in software engineering, whereby existing components ... are used for new purposes and combined to support new capacities, without disrupting their participation in existing programs” (Anderson, 2007, p.13). This is the theoretical motivation behind our generic model. Based on this we believe that it is useful to examine how cognitive modules can be modified and re-purposed to perform different cognitive functions.

The Python ACT-R system supports this exploration by allowing the modeller to customize the ACT-R architecture. In the current Lisp ACT-R system, the underlying modules can certainly be modified, but this requires delving into the underlying implementation of the architecture. In Python ACT-R, creating memory systems and customizing them in various ways is a basic part of the modelling process (much like defining productions and chunks). No understanding of the underlying implementation is needed. This makes architectural exploration available to any cognitive modeller, not just those willing to implement models from scratch.

Salience

We began this project by updating Python ACT-R’s capabilities by implementing ACT-R's new visual salience system (Byrne, 2006). This was originally developed to model the “pop-out” effect familiar to vision researchers. This is implemented in Python ACT-R as a separate module that can connect to the visual system. However, because the visual system in Python ACT-R is built using the generic system as the declarative memory system, it was also possible to connect the salience module to declarative memory. However, it should be noted that we are not claiming that the same *physical neurons* in the brain may be responsible for both visual and memory effects. Rather, we believe that a similar cognitive structures may be involved, and that a common computational model may be used for both.

To develop this idea, we first describe the ACT-R declarative memory system and the vision system. Next, a generic system is described that is capable of being customized via particular sub-components to perform either task. We then demonstrate its capabilities, showing particular differences between this vision model and the standard ACT-R vision model. Finally, we investigate what capabilities are gained by the declarative memory system now that the visual salience system can be integrated with it. The result is both a novel model of *distinctiveness* in memory and a demonstration of the usefulness of cognitive redeployment.

Declarative Memory

The ACT-R declarative memory system consists of a module for storing *chunks* (small meaningful collections of data, such as "Fido is a dog") and a retrieval buffer for storing the currently recalled chunk. Chunks are placed into memory whenever they are used by an ACT-R production rule. A production rule can also make a request from memory, asking for a chunk that fits a particular pattern (such as "Fido is a *what?*"). If a chunk is found, then its contents are placed in the retrieval buffer.

In many cases, there will be several chunks that match the requested pattern. In these situations, the chunk with the highest *activation* is retrieved. The activation of a chunk is based on a variety of factors, the most important of which is the *base level learning* equation. Here, a chunk's activation is increased when it is used, and this activation decays over time. There is also a formula for adjusting the activation of chunks that almost match the memory request pattern (partial matching), a formula for increasing the activation of chunks that are similar to chunks already in buffers (spreading activation), and a random amount of noise that leads to variability in behaviour.

However, there is no way of *modifying* chunks once they are placed in memory. Therefore, to avoid the problem of repeatedly retrieving the most active chunk in situations where this is not appropriate, ACT-R has a system for maintaining "Fingers of Instantiation" (or FINSTs). This allows a memory request to indicate that it should not recall a recently retrieved item. Details and formulas for the declarative memory system can be found in (Anderson & Lebiere, 1998).

Vision

The vision system in ACT-R (formerly a part of ACT-R/PM) has two sub-systems and two buffers associated with these sub-systems. These two sub-systems are the "where" and "what" systems, where the former is in charge of directing visual attention, and the later in charge of determining information about the object at the attended location.

To perceive an object, an ACT-R production is first used to instruct the vision system to find a location in space. This can be something as simple as "find any object" or as complex as "find a red object on the right side of the screen that's above the object currently being attended to". When an object is found that matches these criteria, the location of the object is placed in the visual location buffer. Next, the "what" system can be used to attend to that location and get information about the object. In ACT-R, this does not involve a theory of visual processing. Instead, the features that have been attached to the object by the modeller are placed in the visual buffer. For example, a picture of a house might be set to create a chunk in the visual buffer that simply says "house".

The vision system also supports a number of other behaviours. If an object moves while being attended to, then attention will follow that object (object tracking). If nothing is being attended to, but a new item appears which matches the previous search request, then it will automatically be attended to (buffer stuffing). Requests can also be made to only attend to "new" or "old" objects (i.e. objects that have recently appeared or been visible for a longer period of time). This functionality is similar to FINST system in declarative memory.

One of the most recent advances in the ACT-R vision system is the addition of a theory of *visual salience* (Byrne, 2006). Here, when an object is being searched for, the system no longer chooses randomly among matching items. Instead, the visual features of the object are examined, and objects that have more *rare* features are more likely to be attended to. This causes the well-known pop-out effect to occur, e.g., where a single red object in a set of blue objects is much more likely to be attended to.

Generic Chunk Storage and Retrieval

Both the declarative memory system and the vision system operate according to the same basic premise: requests to find chunks conforming to a particular pattern are made, and the result is placed into an associated buffer (note, we are treating objects in the visual field as chunks here). This is the basic definition of a *content-addressable* memory system, and has a simple implementation: chunks are placed in the memory, these are examined individually to see which ones match, and one of these is chosen as the final result.

The key question is what to do when multiple chunks match. In declarative memory, the base level learning, partial matching, and spreading activation systems (as well as random noise) all combine to form an *activation* value, and the chunk with the highest activation value is returned. The visual system can be thought of as operating in a similar way, with the activation of the chunks in the visual field boosted according to their level of salience.

Our approach is to treat the different process involved in retrieval as *sub-modules*. Without any sub-modules, the activation value of all chunks in the generic memory system is considered to be zero. Since all the chunks have the same activation value, if more than one chunk matches, the system chooses randomly between them.

As part of the model creation process, Python ACT-R allows the modeller to attach particular sub-modules to adjust the activation values. This is similar to, but more flexible than, the standard ACT-R approach of turning on or off the various aspects such as partial matching or spreading activation. The following sub-modules have been written:

Base Level: the standard ACT-R base level learning equation. Activation increases whenever a chunk is added into memory, and decays over time. This also includes the new optimized version (Petrov, 2006).
Spreading Activation: increases activation for slots that have chunk values matching those in particular buffers. Allows for configuring which buffers are used and the strength of the spreading.
Partial Matching: allows chunks that do not exactly match the request to be returned, but at a decreased level of activation. Note that this sub-module requires a slightly more complex integration with the memory module, as it not only adjusts the activation levels, but also adjusts the set of chunks that could be returned.
Noise: a configurable amount of logistic noise added to the activation of each chunk. Can also add a random fixed amount to a chunk when it is first created. Equivalent to the Lisp ACT-R parameters ANS and PAS.
Salience: increases the activation of chunks based on the rarity of the slot values. Requires specifying a context of which chunks to consider when determining how rare particular values are.

To complete the implementation, the chunk storage system also defines a *threshold* (a value that activations must be above in order to be retrieved) and a *latency* (a value determining how lower activations require more time to recall). Furthermore, a FINST (fingers of instantiation) system is also available, which keeps track of what chunks have been recently recalled, and allows those chunks to be temporarily ignored.

The first four of these sub-modules should be familiar to ACT-R users. They form the terms in the standard ACT-R activation equation. The fifth sub-module implements the new salience estimation process (Byrne, 2006). With this approach, the declarative memory system and the vision system can constructed from the same basic system.

Vision as Memory

The core part of a vision system can now be implemented via the new generic system. For the most part, this generic system will be familiar to those who have worked with the ACT-R declarative memory system. Using this for vision requires a few special considerations. Chunks representing the currently visible objects are stored in a generic memory system that can be thought of as representing the visual trace. The ACT-R model can then make requests of that system to find particular objects, just as is done in the standard ACT-R vision system. However, there are two major complications to this process.

First, there needs to be a way to fill the memory with the currently available chunks. For this type of function, Python ACT-R also allows for the creation of *separate production systems*. These can be specified in exactly the same way as the core production system, allowing this sort of sub-module to be easily implemented. This separate production system for vision can also be used to implement buffer-stuffing and tracking (see West, Stewart, Pyke, & Emond, 2006 for an example of using this approach for buffer stuffing).

We have found that the production system for filling the visual memory requires a very short action time (the time required for a production to fire), around 5 to 10 msec in order to be reasonably responsive to changing stimuli. Of course, since this is a separate production system it, does not affect how quickly the productions in the ACT-R procedural memory fire (normally 50 msec). Furthermore, if this system uses the Base Level sub-module and a high threshold (~5.5) and decay (~0.95), then it is not necessary to also clear the memory: the decay of activation for old chunks will automatically ensure objects that no longer are visible are not returned. This can be seen in Figure 1.

One difference between this system and the standard ACT-R vision system is that there is no bias towards returning *new* objects. Instead, Figure 1 shows that older objects can have a higher activation if they have been in the visual field for a longer amount of time. However, this effect disappears if chunks that were in the visual field on the previous cycle are included in the context set for determining salience. In this case, new objects will be returned if they a sufficiently salient.

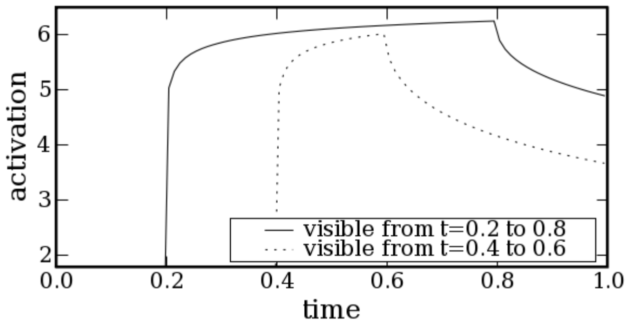

Figure 1: Activation levels of two chunks in the new vision system, visible for different periods of time. Activation uses the Base Level learning equation with d=0.95.

The second complication involves the separate "what" and "where" systems in ACT-R vision. One approach is to ignore this distinction, and not model the vision system to this fine a degree. This is the approach taken by the SOS vision system for ACT-R (West & Emond, 2002), and has been shown to be useful in situations where the details of vision are not an important part of the cognitive process being modelled.

However, an alternative approach is to implement vision using two separate memory systems: one that is filled with the *locations* of all available objects. This can be used to implement the "where" system, and results in filling a visual location buffer with a location that can be attended to. The "what" system can then be a separate memory, which is only filled with items near the currently attended visual location. This maps well onto the behaviour of the current ACT-R vision system.

In either case, the result is a memory-based system that constantly maintains a collection of chunks representing the visible objects available to the ACT-R model. As objects appear and disappear, the activation levels of chunks in this visual system change. When a retrieval request is made (i.e. when ACT-R chooses to attend to a visual object), a chunk representing that object is placed into a visual buffer. The activation levels of the chunks control this process just as they do in ACT-R declarative memory. The result is a variation of the generic chunk storage system that functions as a visual system.

Applying Memory Sub-Modules to Vision

Implementing vision via the standard ACT-R memory system can, by itself, be seen as a useful modelling advance, as it provides a parsimonious explanation of both systems. Having both vision and memory built using the same cognitive components shows how it is possible to redeploy cognitive facilities from one task to another. Of course, significant work is still needed to rigorously compare the performance of this new vision system, the ACT-R vision system, and real human vision. This work is on-going.

However, it is worth pointing out two significant advantages that are available to this new vision system due to its integration with existing memory models. Just as the Base Level sub-module was used to implement item decay from visual memory, it is also possible to use both the Partial Matching sub-module and the Spreading Activation sub-module.

Applying Partial Matching to vision provides a natural implementation of many of the special-case features that are used in the current ACT-R vision system. Instead of the VISUAL-MOVEMENT-TOLERANCE parameter, which indicates how far away from a point an object can be and still be noticed, the partial matching system can gradually reduce the activation of chunks farther away from the point of interest. A similar approach can be taken for features such as colour, where a search for a *red* should also find objects that are *pink* (although perhaps at a slightly lower activation level).

Another new possibility arises with Spreading Activation. Here, the activation of visual chunks can be increased based on the current contents of the goal buffer (or any other buffer). This can be seen as a type of top-down processing influencing visual attention. This aspect of top-down and bottom-up salience is discussed in more detail at the end of this paper.

A detailed analysis of these possibilities is still in progress. However, the mere fact that these options appear naturally from this method of modelling vision is promising in and of itself.

Memory as Vision

It is also possible for us to apply the salience module used in the vision system to the declarative memory system. When used in the context of vision, this module leads to pop-out effects. If this same system is applied to declarative memory, then it results in a bias towards recalling unique chunks. This can be seen as an implementation of *distinctiveness*. Chunks that have similar slot values will tend to have lower activations than chunks that have distinctive values. This makes it easier to recall rare or special items in memory (a well known memory effect).

We now turn to a specific example of the modelling capabilities gained by using the salience module to create a distinctiveness effect in declarative memory.

Release from Proactive Interference

The Salience sub-module increases the activation of chunks with unique slot values. This can be seen as functionally equivalent to decreasing the activation of chunks with similar slot values. That is, if a chunk is similar to (i.e. is semantically related to) another chunk in memory, then the activation for those chunks will be less than it would be otherwise.

This precisely corresponds to the classic phenomenon of release from proactive interference. Here, a list of words is presented to a subject, and they are then asked to recall as many as possible (usually with a distractor task to eliminate rehearsal). Four groups of words are usually presented. The first three groups all contain words of a similar category (e.g., household items). The fourth group, however, presents words from a different category (e.g., animals). The observed effect is that recall accuracy decreases over the first three groups, and then increases for the fourth, although it does not usually increase up to the accuracy of the first group.

To implement this in ACT-R, the chunks representing words are assumed to have a slot indicating what category they are in. The model uses the Salience, Noise, and Base Level sub-modules, as well as the FINST system (to stop the model from recalling the same word repeatedly). Results from the model are shown in Figure 2.

These results show the general pattern seen in human data. The exact shape of this curve is highly dependent on a large number of factors (Hasher et al., 2002), so we have not attempted to fit this data to a particular set of results. The model currently has enough free parameters to adjust to fit any similar shape, so further data is required to constrain the model.

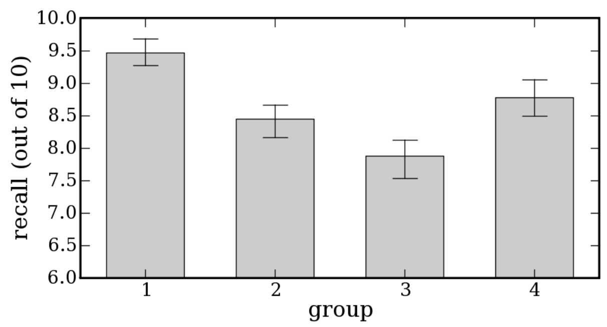

Figure 2: Recall accuracy for the Release from Proactive Interference task. The words presented in groups 1-3 are all from the same category, while group 4 is different.

In particular, one issue is the *context* for determining how rare particular slot values are. Clearly, not *all* chunks in declarative memory should be considered. Instead, we are examining the possibility of limiting the chunks to only those above a certain activation, or weighting them by their activation.

Salience and Spreading Activation

In examining the effects of these modules in various configurations, there is a certain similarity between the Salience sub-module and the Spreading Activation sub-module. Both of them are used to increase activations of particular chunks (i.e. to make them more or less likely to be recalled) based on contextual information. For the Salience system, this increase is based on the uniqueness of the chunk among some set of chunks. For Spreading Activation, this increase is based on the similarity between the chunk and some specific buffer content.

One way of interpreting this difference is to consider Salience to be a model of *bottom-up* attention, while Spreading Activation is *top-down* attention. That is, Salience is a low-level, highly automatic process that is only somewhat controllable by high-level reasoning. It may be possible to adjust the context used to determine Salience (for example, by only considering objects in the left half of the visual field, (c.f. Byrne, 2006), or only considering chunks of a particular type), but it is not organized for fine-grained control.

In contrast, the Spreading Activation system can be used to focus attention on chunks based on their similarity to one specific focused chunk. Here, chunks that are related are connected in a semantic web, with shared chunk values allowing attention to one chunk to increase the activation of chunks that are connected to it. Applying this to vision allows for a similar focusing of attention on objects related to the current topics of thought.

It is also interesting to note that both sub-modules share a similar equation, based on the logarithm of one over the number of chunks which share that slot value. Indeed, it is possible to see Salience as a version of Spreading Activation that spreads from *every possible* chunk (or every chunk in the context set if this set is defined in some way), rather than from one particular chunk. This suggests that these two sub-modules may share a common underlying implementation as well.

Conclusion

Inspired by the idea of cognitive redeployment, we have been exploring how the components of ACT-R can be adapted to perform different cognitive tasks. This architectural flexibility has been an important part of our Python ACT-R project, allowing for the cognitive architecture itself to be adjusted. This capability was exploited here to show that two disparate systems (vision and declarative memory) may share common underlying components.

In addition, the commonalities between vision and declarative memory allow us to take particular features from vision (or memory) and apply them to memory (or vision). If the visual salience system is connected to declarative memory, we find a natural implementation of the distinctiveness phenomenon, leading to a novel model of release from proactive interference. Furthermore, salience and spreading activation seem to form a complementary pair; one implementing bottom-up attention, and the other implementing top-down attention.

References

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

Anderson, M. (2007). Evolution of cognitive function via redeployment of brain Areas. *Neuroscientist*, 13(1):13–21.

Anderson, M. (forthcoming). The massive redeployment hypothesis and the functional topography of the brain. *Philosophical Psychology.*

Byrne, M. (2006). *A theory of visual salience computation in ACT-R.* 13th Annual ACT-R Workshop, Pittsburgh, PA.

Hasher, L., Chung, C., May, C., & Foong, N. (2002). Age, time of testing, and proactive interference. *Canadian Journal of Experimental Psychology*, 56(3), 200-207.

Petrov, A. A. (2006). *Computationally efficient approximation of the base-level learning equation in ACT-R.* 7th International Conference on Cognitive Modeling. Trieste, Italy.

Stewart, T.C. & West, R. L. (2006) *Deconstructing ACT-R.* 7th International Conference on Cognitive Modelling. Trieste, Italy.

Stewart, T.C. & West, R.L. (in press) Deconstructing and Reconstructing ACT-R: Exploring the Architectural Space. *Cognitive Systems Research.*

West, R.L, & Emond B. (2002). *SOS: A simple operating system for modeling HCI with ACT-R.* 7th Annual ACT-R Workshop. Pittsburgh, PA.

West, R. L., Stewart, T. C., Emond, B., & Pyke A. (2006). *Modelling emotion in ACT-R.* 13th Annual ACT-R Workshop, Pittsburgh, PA.