ACT-R Software Updates
August-December 2008

Dan Bothell
Overview

• Notable changes since the Summer release (r617)
  – Fall release in October r696
  – Current (Winter) release is r723

• All updates are available
  – Text log
    http://act-r.psy.cmu.edu/~webcron/actr6log.txt
  – RSS feed
    http://act-r.psy.cmu.edu/~webcron/actr6feed.xml
Outline

- Changes available in the Fall release [r696]
- Changes available in the Winter release [r723]
Fall Release Highlights

• Environment tools
• Updates to existing modules
• New module
• Performance
• Miscellaneous
Environment Changes

• Graphic trace tools no longer require setting the :save-buffer-trace parameter to t
  – Just have to open the window prior to the model run, but after any reset
  – Setting the parameter still works and is safe across resets

• Chunks and productions in the graphic traces can be clicked on to open the declarative and procedural viewers with the item selected
Environment Additions

- Graphic run histories
  - Productions
  - Retrievals
  - Buffers

- Not available by default
  - In extras/history-tools
  - Move the files as indicated in the readme.txt
Production history
Production history (cont)

• One row per production
• Column for every conflict-resolution labeled with the time it occurred
• Color indicates if the production was in the conflict set
  – Green chosen production
  – Orange in the set and not chosen
  – Red not in the set
• Putting the cursor over a block shows
  – Utility for productions in the set
  – Whynot info for those not in the set
Retrieval history

Matching Chunks

<table>
<thead>
<tr>
<th>Times</th>
<th>p1</th>
<th>p2</th>
<th>p3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.235</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.485</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.585</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Details

P1
ISA COMPREHEND SENTENCE
RELATION IN
ARG1 HIPPIE
ARG2 PARK

Declarative parameters for chunk P1:
:Activation 0.214
:Permanent Noise 0.000
:Base-Level 0.000
:Source-Spread 0.214
:Sjis ((P1 . 1.6) (IN . -1.0390574) (HIPPIE . 0.21370566) (PARK . 0.21370566))
:Last-Retrieval-Activation 0.214
:Last-Retrieval-Time 0.585

Request

ISA COMPREHEND-SENTENCE
ARG1 HIPPIE
Retrieval history (cont)

- Left column is times when a retrieval request occurred
- Right column is all chunks which matched the request at the time selected in the left column
- Windows on the right show details
  - Top window
    - The chunk selected in the middle column with parameters at the time of the request
  - Bottom window
    - The retrieval request which was made
Buffer history

The image displays a buffer history interface with a list of buffers and their details. The buffers are listed under the `Times` column, and the `Details` column shows the properties of the selected buffer, such as `.lastIndexOf`, `hasLocation`, `fntText`, and `height`. The buffer history includes entries like `retrieval`, `manual`, `goal`, and `visual-location`, among others.
Buffer history (cont)

- Left column shows all the times when some scheduled buffer change occurred
- Right column shows all the buffer names
- Window on the right shows details of the selected buffer at the selected time
  - Chunk in the buffer at that time
  - Buffer status information at that time
Fall Release Highlights

- Environment tools
- Updates to existing modules
- New module
- Performance
- Miscellaneous
Declarative module

• New parameter :w-hook
  – Allows one to adjust the $W_{kj}$ values in the spreading activation equation
  – Set to a function like other hooks
  – Passed two parameters:
    • buffer name and slot name
  – If it returns a number that overrides the default $W_{kj}$ value
Vision Module

• New queries for visual buffer
  – Scene-change
  – Scene-change-value

• Alternate mechanism for detecting screen changes
  – More reliable than visual-location buffer stuffing
  – Has a settable change threshold
    • :scene-change-threshold
Scene-change

• The query:
  `?visual>
  scene-change t
will be true when all of these are true
– there has been a proc-display call within :visual-onset-span seconds
– The change in the visicon at that time was at or above the threshold (.25 default value)
– The notice has not been explicitly cleared
Scene-change (cont)

• Change is defined as:

\[
Change = \frac{d + n}{o + n}
\]

- \( o \): The number of features in the visicon prior to the update
- \( d \): The number of features which have been deleted from the original visicon
- \( n \): The number of features which are newly added to the visicon by the update

• Can be explicitly cleared with a clear-scene-change request or the existing clear request
  +visual> isa clear-scene-change
  +visual> isa clear
Scene-change-value

- Primarily for buffer-status use:
  ```
  VISUAL:
  ...
  scene-change : T
  scene-change-value : 1.0
  ```
- Shows the last change value
- Can be queried with a number:
  ```
  ?visual>
  Scene-change-value x
  ```
- Will be true if last scene-change-value >= x
Fall Release Highlights

- Environment tools
- Updates to existing modules
- New module
- Performance
- Miscellaneous
Blending module

- New module to perform blended retrievals
- Not installed by default
  - In extras/blending
  - See the blending-read-me.txt for installation info and module details
- Also Included
  - Christian’s slides describing the original mechanism in detail
  - Some example models using the module
Blending module overview

• Assumes the default declarative module exists
• Has one buffer called blending
• Takes requests like the retrieval buffer does
  – Results in a chunk being placed into the blending buffer if successful
• Responds to queries for state busy, free and error the same way the retrieval buffer does
  – State is independent of the declarative module’s
Blending request basics

• Create a resulting chunk with
  – The chunk-type of the requested chunk
  – All the explicit values given in the request
  – Blended slot values for all other slots
Blending request details

• Start with the set of chunks which match the request
• Compute the activation, $A_i$, of each chunk in that set using the normal declarative mechanisms
• Use those activations and the temperature setting of the blending module (:tmp) to compute the probability of each chunk in the set being retrieved using the Boltzmann equation
• Now we have a $p_i$ value for each chunk $i$ in the matching set
Blending request details (cont)

• For each blended slot
• Consider the values in the corresponding slots of every chunk $i$ in the matching set
  – Call that $v_i$
• Three cases
  – All $v_i$ are numbers
  – All $v_i$ are chunks
  – Something else
    • Not going to cover this case
All $v_i$ are numbers

• The value for the slot is:

$$\sum_{i} p_i \cdot v_i$$
All $v_i$ are chunks

- Consider all chunks of that chunk-type as potential values (or all chunks if there is no common chunk-type among the $v_i$ chunks)
- For each potential value $j$ compute:

\[ B_j = \sum_i p_i \cdot \text{sim}(i, j)^2 \]

- The value for the slot is the chunk $j$ with the minimum $B_j$ value
Blending time and success

- Compute a match score for the blended chunk
  \[ M = \log \sum_i e^{A_i} \]

- If \( M \geq \) the retrieval threshold the chunk is placed in the blending buffer after a time
  \[ BT = F e^{-(f*M)} \]

- Otherwise it fails after a time based on the retrieval threshold
Blending Example (matching set, $A_i$ and $p_i$)

blending-test-1.lisp:

```
(sgp :v t :blt t :esc t :ans .25 :rt 4)
```

(chunk-type target key value size)

(chunk-type size)

(add-dm
  (tiny isa size) (small isa size) (medium isa size)
  (large isa size)(x-large isa size)

(a isa target key key-1 value 1 size large)
(b isa target key key-1 value 2 size x-large)
(c isa target key key-1 value 3 size tiny)
(d isa target key key-2 value 1 size nil)
(e isa target key key-2 value 3 size small))

(set-similarities (tiny small -.1) (small medium -.1)
  (medium large -.1)(large x-large -.1)(tiny medium -.3)
  (small large -.3)(medium x-large -.3)(tiny large -.6)
  (small x-large -.6)(tiny x-large -.9))

(p p1 ... ==>
  +blending>
  isa target
  key key-1)

0.050 BLENDING START-BLENDING

Blending request for chunks of type TARGET

Blending temperature defaults to (* (sqrt 2) :ans): 0.35355338

Chunk C matches blending request

Activation 3.5325232

Probability of recall 0.2851124

Chunk B matches blending request

Activation 3.763482

Probability of recall 0.5479227

Chunk A matches blending request

Activation 3.3433368

Probability of recall 0.16696489

Slots to be blended: (VALUE SIZE)
Blending Example (computing value slot)

blending-test-1.lisp:

(sgp :v t :blt t :esc t :ans .25 :rt 4)
(chunk-type target key value size)
(chunk-type size)
(add-dm
  (tiny isa size) (small isa size) (medium isa size)
  (large isa size) (x-large isa size)
  (a isa target key key-1 value 1 size large)
  (b isa target key key-1 value 2 size x-large)
  (c isa target key key-1 value 3 size tiny)
  (d isa target key key-2 value 1 size nil)
  (e isa target key key-2 value 3 size small))

(set-similarities (tiny small -.1) (small medium -.1)
  (medium large -.1) (large x-large -.1) (tiny medium -.3)
  (small large -.3) (medium x-large -.3) (tiny large -.6)
  (small x-large -.6) (tiny x-large -.9))

(p p1 ... =>
  +blending>
    isa target
    key key-1)

Finding blended value for slot: VALUE
Matched chunks' slots contain: (3 2 1)
Magnitude values for those items: (3 2 1)
With numeric magnitudes blending by weighted average
Chunk C with probability 0.2851124 times magnitude 3.0 cumulative result: 0.85533726
Chunk B with probability 0.5479227 times magnitude 2.0 cumulative result: 1.9511826
Chunk A with probability 0.16696489 times magnitude 1.0 cumulative result: 2.1181474
Final result: 2.1181474
Blending Example (Computing size slot)

Finding blended value for slot: SIZE
Matched chunks’ slots contain: (TINY X-LARGE LARGE)
Magnitude values for those items: (TINY X-LARGE LARGE)
When all magnitudes are chunks or nil blending based on common chunk-types and similarities
Common chunk-type for values is: SIZE
Comparing value TINY
Chunk C with probability 0.285 slot value TINY similarity: 0.0 cumulative result: 0.0
Chunk B with probability 0.547 slot value X-LARGE similarity: -0.9 cumulative result: 0.443
Chunk A with probability 0.166 slot value LARGE similarity: -0.6 cumulative result: 0.503
Comparing value SMALL
Chunk C with probability 0.285 slot value TINY similarity: -0.1 cumulative result: 0.002
Chunk B with probability 0.547 slot value X-LARGE similarity: -0.6 cumulative result: 0.200
Chunk A with probability 0.166 slot value LARGE similarity: -0.3 cumulative result: 0.215
Comparing value MEDIUM
Chunk C with probability 0.285 slot value TINY similarity: -0.3 cumulative result: 0.0256
Chunk B with probability 0.547 slot value X-LARGE similarity: -0.3 cumulative result: 0.074
Chunk A with probability 0.166 slot value LARGE similarity: -0.1 cumulative result: 0.076
Comparing value LARGE
Chunk C with probability 0.285 slot value TINY similarity: -0.6 cumulative result: 0.102
Chunk B with probability 0.547 slot value X-LARGE similarity: -0.1 cumulative result: 0.108
Chunk A with probability 0.166 slot value LARGE similarity: 0.0 cumulative result: 0.108
Comparing value X-LARGE
Chunk C with probability 0.285 slot value TINY similarity: -0.9 cumulative result: 0.230
Chunk B with probability 0.547 slot value X-LARGE similarity: 0.0 cumulative result: 0.230
Chunk A with probability 0.166 slot value LARGE similarity: -0.1 cumulative result: 0.232
Final result: MEDIUM
Blending Example (time and success)

blending-test-1.lisp:
(sgp :v t :blt t :esc t :ans .25 :rt 4)
(chunk-type target key value size)
(chunk-type size)
(add-dm
  (tiny isa size) (small isa size) (medium isa size)
  (large isa size)(x-large isa size)

(a isa target key key-1 value 1 size large)
(b isa target key key-1 value 2 size x-large)
(c isa target key key-1 value 3 size tiny)
(d isa target key key-2 value 1 size nil)
(e isa target key key-2 value 3 size small))

(set-similarities (tiny small -.1) (small medium -.1)
  (medium large -.1)(large x-large -.1)(tiny medium -.3)
  (small large -.3)(medium x-large -.3)(tiny large -.6)
  (small x-large -.6)(tiny x-large -.9))

(p p1 ... ==>
  +blending>
    isa target
    key key-1)

This is the definition of the blended chunk:
(ISA TARGET KEY KEY-1 SIZE MEDIUM VALUE 2.1181474)

Computing activation and latency for the blended chunk
Activation of chunk C is 3.5325232
Activation of chunk B is 3.763482
Activation of chunk A is 3.3433368
Activation for blended chunk is: 4.6598654
0.050 PROCEDURAL CONFLICT-RESOLUTION
0.059 BLENDING BLENDING-COMPLETE
Fall Release Highlights

- Environment tools
- Updates to existing modules
- New module
- **Performance**
- Miscellaneous
Performance

• Added a set of test models to measure long term performance values
  – Ensure things are at least linear

• Lots of little changes
  – Minor code changes (append -> nconc, etc)
  – Internal representations
    • Things people shouldn’t notice

• Two updates for the vision module
Vision module performance

- Since it uses chunks internally there’re a lot of “garbage” chunks created
  - used once and then not needed
- The built-in GUI tools now reuse and delete their chunks when not needed
- New parameter :delete-visicon-chunks
  - The module’s internal chunks also get deleted
  - Defaults to t
  - May need to set to nil to work with some extensions (EMMA)
Fall Release Highlights

- Environment tools
- Updates to existing modules
- New module
- Performance
- Miscellaneous
Miscellaneous

• Manual now has sections on
  – Working with chunk-specs
  – Accessing and using buffers
  – Adding new chunk parameters
  – Defining new modules

• New command: capture-model-output
  – Works like no-output to suppress output except it stores it in a string which it returns

• Changes to some feature checks to work right with Clozure Common Lisp (was previously OpenMCL)
Updates for the Winter release

- Changed the internal mechanisms used to hold and compute the chunk fan values
- Added a third reset function option for modules
- New option for normalizing chunk names
- Ongoing performance improvement work
  - Still experimental
  - Available for testing if interested
New fan storage mechanism

• Previously
  – Saved the list of all i’s in the chunk j (j is the source i is the chunk with a connection to j)
  – Required searching that list when computing fan_{ji} (fan-out_{j}/fan-in_{ji})

• Now
  – Store only the total fan-out count in j
  – Store the fan-in count for each j in i

• Much faster for models with large fan-outs

• If you were accessing that fan-out list it’s not there now
  – Wasn’t available through the normal mechanisms anyway
  – Could add a flag to still save it if people really need that
Third reset function

• Modules now have an additional option for when they can get called during reset
  – After all the model code evaluated
• It’s set as a third item in the :reset list when defining the module if needed
“New” Chunk name Normalizing

- New parameter :dcnn (dynamic chunk name normalizing)
- Works in conjunction with :ncnar
- When both are true (the default values)
  - Chunk names are normalized as the model runs instead of at the end
  - When chunks merge all slots of ALL chunks which have the merged name are updated to the true name
  - Much closer to how the older ACT-Rs worked
More on :dcnn

• Primarily for model debugging
  – Never see multiple names for one chunk
  – Should not affect the operation of the model
• May or may not be faster that normalizing at the end
  – Depends on how much merging occurs, the interrelations among the chunks, and how many chunks the model has
• Does require extra storage to hold the back-links
  – So a larger memory footprint is required to use it
• For best performance :ncnar should still be set to nil
  – Disables all the normalizing
Simple :dcnn example

(chunk-type goal slot)

CG-USER(12): (sgp :dcnn nil)
(NIL)

(add-dm (name isa chunk))

CG-USER(13): (run 10)

(p start
  ?goal> buffer empty
  ==> +goal> isa goal
  +retrieval> isa chunk)

THE VALUE IS NAME-0

(p set-up
  =goal> isa goal
  =retrieval> isa chunk
  ==> =goal>
    slot =retrieval)

(p report
  =goal>
    isa goal
  slot goal
  slot =value
  ==> !output! (the value is =value)
      !stop!))

THE VALUE IS NAME

CG-USER(9): (sgp :dcnn t)
(T)

CG-USER(10): (run 10)

0.050 PROCEDURAL  PRODUCTION-FIRED START
0.050 GOAL  SET-BUFFER-CHUNK GOAL GOAL0
0.050 DECLARATIVE  SET-BUFFER-CHUNK RETRIEVAL NAME
0.100 PROCEDURAL  PRODUCTION-FIRED SET-UP
0.150 PROCEDURAL  PRODUCTION-FIRED REPORT

0.150 ------ BREAK-EVENT Stopped by !stop!

0.050 PROCEDURAL  PRODUCTION-FIRED START
0.050 GOAL  SET-BUFFER-CHUNK GOAL GOAL0
0.050 DECLARATIVE  SET-BUFFER-CHUNK RETRIEVAL NAME
0.100 PROCEDURAL  PRODUCTION-FIRED SET-UP
0.150 PROCEDURAL  PRODUCTION-FIRED REPORT

THE VALUE IS NAME

0.150 ------ BREAK-EVENT Stopped by !stop!
Ongoing Update Work

• Improve performance
  – Development focus has been primarily on functionality up to this point
  – Try to stay ahead of demand

• Identify a mechanism that
  – Takes a significant amount of time
  – Common to most/all models
  – Shows an opportunity for improvement without significantly affecting current users
The conflict-resolution event

• Common to just about every model
• Profiling a variety of models showed that it typically accounts for somewhere between 20-50% of the run time
  – Primarily in the production matching code
    • Not the actual “conflict resolution” calculation
• Matching is a completely internal mechanism
  – No user hooks or access to the low-level operation
Current Algorithm

• For each production
  Test each condition until
    all successful or one fails
    – Same as previous versions*

• Linear in the total number of conditions in the productions
  – Because each condition is a simple test – no search

• Checked with a simple performance testing model
  – Each production has two conditions and no actions
  – One production matches and n do not match
    • (p target =goal> isa test slot target ==>)
    • (p failn =goal> isa test slot failn ==>)
  – No other events in the model
  – Essentially the only thing happening is conflict-resolution events
• Linear – good
• Doesn’t seem too bad
• Compare to similar model in ACT-R 5.0
Run for 1000 simulated seconds

Number of productions vs. Real time in seconds for ACT-R 6 [r690] and ACT-R 5.
• Room for improvement
• Not unexpected
  – Christian did a lot of work to improve ACT-R 4.0
  – ACT-R 6.0 has had little performance work
  – More abstraction in 6.0
    • Probably never get to performance of 4.0/5.0 using the same algorithm
Plan forward

• Two strategies
  – Improve the internal production representation and code
  – Change the algorithm

• Working on both basically in parallel
Code changes

• Two big updates so far
• Replaced the use of the general chunk matching commands with specific code for buffer chunks which cache the results
• Replaced the lambdas that were generated at parse time with a structured representation that’s tested at run time instead of evaled
Run for 1000 simulated seconds

Number of productions vs. Real time in seconds for different ACT-R versions:
- ACT-R 6 [r690]
- ACT-R 5
- ACT-R 6 [r705]
- ACT-R 6 [r719+]
Algorithm

- Better than linear across all conditions?
- Why not use RETE?
  - Doesn’t really fit our situation
    - We don’t require search in matching
      - Already linear in number of conditions
    - We have a fairly small set of items to match (buffer slots)
- Try just a simple decision tree
Decision tree

• Only considering the constant tests in the productions at this point
  – Isa tests, specific slot values, and queries

• Nodes represent the conditions
  – Branches for the possible values

• Leaves are a set of productions which may need further testing

• At matching time it just needs to walk a path from the root to a leaf
Current implementation

• Creates the tree given all the productions
  – That’s why the third reset hook was added
• Use an existing algorithm to build it – ID3
  – Add the condition which has the most information gain
    – Heuristic favors smaller depth trees
• Stop a branch when there’re no more common conditions to test
Run for 1000 simulated seconds

Number of productions

Real time in seconds

ACT-R 6 [r690]
ACT-R 5
ACT-R 6 [r705]
ACT-R 6 [r719+]
ACT-R 6 [tree]
Needs more testing

- That test model is essentially the best possible situation for the tree
- Run times with other models did improve
- Not without potential issues
  - Time to build the tree
  - Space to hold the tree
- Not enabled by default
  - need to set the :use-tree parameter to t
  - Should work for all models including those using production compilation