Chapter 1: Modeling paradigms in ACT-R

Niels Taatgen, Christian Lebiere and John Anderson, Carnegie Mellon University

1. Introduction

The goal of this chapter is to discuss the ACT-R cognitive architecture (Anderson et al., in press) from both a theoretical and practical perspective. Instead of the more traditional approach of presenting representations, performance mechanisms and learning mechanisms (e.g. Anderson & Lebiere, 1998), the architecture will be discussed along the lines of modeling paradigms, methods by which the architecture can be used to model specific phenomena. These modeling paradigms serve as templates and building blocks for cognitive models, including models of multi-agent interaction, of which examples will be presented in later chapters. The version of the architecture discussed here is ACT-R 5.0. Central to ACT-R is the notion of a declarative memory for facts, and a procedural memory for rules. ACT-R 5.0 extends this basis with a set of modules that interact with the outside world. As a consequence, declarative memory has become another module, while the production system takes the center position, connecting all the modules together (figure 1).

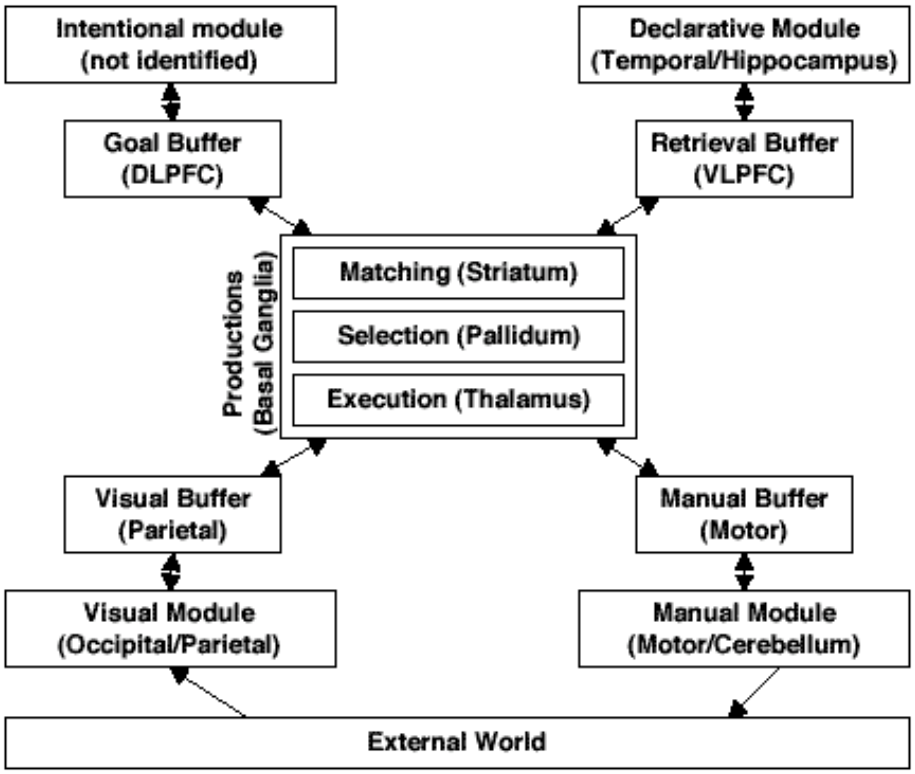


Figure 1: Overview of ACT-R 5.0

The production system does not have unlimited access to the various modules, but can only communicate with them through buffers. For example, in order to retrieve a certain fact from declarative memory, a request has to be made to declarative memory in the form of a partially completed pattern. The declarative module will then try to complete the pattern, after which the result is placed back in the retrieval buffer, where it can be matched and used by another rule. Production rules in ACT-R therefore serve a switchboard function, connecting certain information patterns in the buffers to changes in buffer content, which in turn trigger operations in the corresponding modules. Production rules in ACT-R do not have the same representational power (and the associated computational problems) as classical production systems. The different

modules in the architecture operate asynchronously, and only communicate through the buffers. Behavior within a module is largely serial, for instance, the declarative model can only retrieve one item at a time, and the visual system can only focus its attention on one item in the visual field at a time.

Apart from the distinction between declarative and procedural memory, rational analysis is a theoretical foundation of ACT-R. According to rational analysis, each component of the cognitive system is optimized with respect to demands from the environment, given its computational limitations. A consequence of this choice is that truth is not a fundamental notion in ACT-R (contrary to systems based on logic), but rather a derivative: useful knowledge is usually true, although true knowledge is not necessarily useful. ACT-R is a so-called hybrid architecture, in the sense that it has both symbolic and sub-symbolic aspects. We will introduce these informally.

Items in declarative memory, called chunks, have different levels of activation to reflect their use: chunks that have been used recently or chunks that are used very often receive a high activation. This activation decays over time if the chunk is not used. Activation also includes a component that reflects the degree to which this chunk matches the current context, as well as a noise component that makes activation a stochastic quantity, and retrieval a probabilistic process. Activation represents the probability (actually, the log odds) that a chunk is needed and the estimates provided for by ACT-R's learning equations represent the probabilities in the environment very well. The level of activation has a number of effects. One effect of activation is that when ACT-R can choose between chunks, it will retrieve the chunk with the highest activation. Activation also affects retrieval time, and whether the chunk can be retrieved at all. The

higher a chunk's activation, the faster it can be retrieved, and the more likely that activation is to be above a retrieval threshold. Chunks cannot act by themselves, they need production rules for their application. In order to use a chunk, a production rule has to be invoked that requests it from declarative memory, and another one that harvests it from the retrieval buffer and does something with it. Since ACT-R is a goal-driven theory, chunks are usually retrieved to achieve some sort of goal.

The selection of production rules is also governed by the principle of rational analysis. Each production rule has a real-valued quantity associated with it called utility. This utility is calculated from estimates of the cost and probability of reaching the goal if that production rule is chosen. The unit of cost in ACT-R is time. ACT-R's learning mechanisms constantly update the parameters used to estimate utility based on experience. If multiple production rules are applicable for a certain goal, the production rule with the highest utility is selected. In both declarative and procedural memory, selections are made on the basis of some evaluation, either activation or utility. This selection process is noisy, so the item with the highest value has the greatest probability of being selected, but other items get opportunities as well. This may produce errors or suboptimal behavior, but also allows the system to explore knowledge and strategies that are still evolving.

In addition to the learning mechanisms that update activation and utility, ACT-R can also learn new chunks and production rules. New chunks are learned automatically: each time a goal is completed or a perceptual/motor event is registered, it is added to declarative memory. If an identical chunk is already present in memory, these chunks are merged and their activation values are combined. New production rules are learned

through the mechanism of production compilation, which combines two rules that fire in sequence into a single rule.

We will present ACT-R using five modeling paradigms. Instance learning uses previous experiences to guide choices, and focuses on ACT-R's declarative memory and partial matching mechanism. In Competing Strategies several strategies compete to solve a problem. ACT-R's utility learning mechanism will ensure that the strategy with the best probability of success for the lowest costs will be used most often. When studying Individual Differences architectural global parameters are identified that correlate with traits in individuals. Models that incorporate Perceptual and Motor Processes use the interaction with the outside world as an additional constraint on behavior. Finally, Specialization of Task-independent Cognitive Strategies allows ACT-R to learn task-specific rules on the basis of general strategies, including interpretation of instructions in declarative memory.

2. Instance Learning

Instance learning or instance theory, originally developed by Logan (1988), is a simple but powerful modeling strategy. The basic idea is that as we solve problems, we store the solutions in memory, and will be able to retrieve these as examples for future problem solving. For example, in tasks where participants have to solve alphabet arithmetic problems, like D + 3 = ?, they initially use a counting strategy. Evidence for this is the fact that the solution time increases linearly with the second addend. However, once participants become more experienced, they are able to retrieve answers for memory,

which is much faster, and the linear increase of solution time with the second addend disappears.

It is easy to model instance learning in ACT-R. Achieved goals are already automatically stored in declarative memory. Items in declarative memory have an activation value that decays over time, lowering the probability of correct recall, and is boosted by additional rehearsals, increasing the probability of recall. Another aspect of instance learning in ACT-R, which is not covered by Logan's original theory, is that the retrieval attempt can produce an example that is slightly different from the example that is searched for. In the case of D + 3 = ?, we might retrieve B + 3 = E instead, because B is similar to D, which would lead to an error. In many other areas requiring less precision than arithmetic, however, an example that is similar to the goal can produce a useful answer, or can serve as a basis for analogy.

Activation in declarative memory

The activation of a chunk determines whether or not it can be retrieved, and if it is retrieved, how much time this takes. As has already been stated, activation reflects past use of a chunk, and its association with the current goal context. The activation of a chunk is the sum of a base-level activation, reflecting its general usefulness in the past, and an associative activation, reflecting its relevance to the current context. The activation of a chunk *i* is defined as:

$$A_i = B_i + \sum_j W_j S_{ji}$$ **Activation Equation**

where B_i is the base-level activation of the chunk *i*, the W_j's reflect the attentional weighting of the elements that are part of the current goal, and the S_{ji}'s are the strengths of association from the elements *j* to chunk *i*. The activation of a chunk controls both its probability of being retrieved and its speed of retrieval. In the case where there are multiple candidates for retrieval, the chunk with the highest activation has the highest probability of being retrieved.

Base-level activation (B_i) rises and falls with practice and delay according to the equation:

$$B_i = \ln\left(\sum_{j=1}^{n} t_j^{-d}\right) \qquad \textbf{Base-Level Learning Equation}$$

t_j is the time since the jth practice of an item. This equation is based on the rational analysis of Anderson and Schooler (1991) studying how the pattern of past occurrences of an item predicts the need to retrieve it. They found that the above equation reflects the log odds an item will reoccur as a function of when it has appeared in the past. In ACT-R, it is assumed that base-level activation would track log odds. Each presentation has an impact on odds that decays away as a power function (producing the power law of forgetting) and different presentations add up (producing the power law of practice—see Anderson, Fincham & Douglass, 1999). In the ACT-R community 0.5 has emerged as the default value for the time-based decay parameter d over a large range of applications.

There are two equations mapping activation onto probability of retrieval and latency. In order to be retrieved, a chunk has to have an activation higher than other chunks that also match the request, and the retrieval threshold τ. Because noise is added to the chunk

activations, a given activation corresponds to a certain probability of recall according to the following equation:

$$P(i) = \frac{e^{A_i/s}}{e^{\tau/s} + \sum_k e^{A_k/s}}$$

Probability of Retrieval Equation

where s controls the noise in the activation levels and is typically set at about 0.25 and k ranges over all the chunks that match. Time to retrieve the chunk is given as

$$T_i = Fe^{-A_i}$$

Latency of Retrieval Equation

where ***F*** is a scaling constant mapping activation to (real) time. Given the mechanism of activation, ACT-R is able to predict under what circumstances an instance will be retrieved, and under which circumstances it is not. An example of such a model (Anderson & Lebiere, 1998, chapter 4) is a slight variation on alphabet-arithmetic done by Zbrodoff (1995) where participants have to judge whether an addition with letters and a number is correct, for example "F+3=I?". By varying the addend in the addition (from +2 to +4), they were able to show that at some point people shifted from using counting, where the time to decide increases linearly with the addend, to memory retrieval, where the time to decide is independent of the addend.

Partial matching in instance retrieval

A slightly more general version of instance retrieval is one in which partial matches are allowed besides exact matches. This is not so useful in alphabet-arithmetic, but in many other tasks an example that is similar to the current goal is useful if an exact example is not available. ACT-R handles partial matching by decreasing the activations of chunks

that do not exactly match. This deduction is smaller if the two mismatched values are more similar. If the model tries to retrieve the answer to "F+3=?", then the fact "F+2=H" would only be mildly penalized, because 2 and 3 are similar. The formula for calculating activation while taking into account mismatches now becomes:

$$A_i = B_i + \sum_j W_j S_{ji} + \sum_k P_k M_{ki}$$ **Activation Equation**

In this equation, M_{ki} represents the mismatch between the requested value and the retrieved value, which can vary between 0 (no mismatch, so no penalty) and -1 (complete mismatch). P_k represents the penalty that is deducted from the activation in case of a complete mismatch. In case of a complete mismatch the full penalty is applied, but when the requested value and retrieved value are similar, only a partial penalty is given.

Example model: Sugar Factory

An example of a model where mismatching examples are indeed useful is by Wallach (Taatgen & Wallach, 2002) of an experiment by Berry and Broadbent (1984) in which participants have to control a system called sugar factory. Each trial in the experiment represents a day in which participants have to decide on the size of the workforce (W, between 1 and 9). They are then told the output of the factory for that day (O, between 1 and 12 tons), and are asked the size of the workforce for the next day. The output of the factory not only depends on the size of the workforce, but also on the output of the previous day, and a random factor of –1, 0 or 1, according to the following equation:

$$O(t) = 2W(t) - O(t-1) + random(-1,0,1)$$

If the output is outside the 1...12 range, it is set to the nearest boundary, 1 or 12. While the output increases linearly with the number of workers, it also decreases linearly with the previous day's output, a somewhat counterintuitive relation. Participants were given the goal of bringing the output to 9 tons of sugar per day, and keeping it at that level. Berry and Broadbent found that participants improve their behavior in this experiment with experience, but are not able to explain the relationship between workers and output after they are done with the task. Wallach's model is therefore based on instances, because instance retrieval can improve performance without the model having an explicit representation of what the rule is. The model stores each experience as a separate chunk in declarative memory, for example:

Transition1239

Isa sugar-goal

Previous-output 3

Workers 8

Output 12

This chunk encodes that a previous output of 3 tons of sugar and a workforce of size 8 resulted in a new output of 12 tons of sugar. In order to determine a workforce for a new day, the model starts with a goal like:

Transition1252

Isa sugar-goal

Previous-output 7

Workers ?

Output 9

This goal represents that yesterday's output was 7 tons of sugar, and the desired target is 9 tons of sugar. In order to determine the workforce, the model will try to retrieve an experience from declarative memory that matches both the previous output and the new output. To this end it needs the following two rules, which will be represented in a pseudo-English representation, with variables in italics:

Retrieval-request-rule

IF the goal is to determine the number of workers to achieve output G and
the output of the previous day was O

THEN send a request to declarative memory for an instance with previous output O
and output G

Retrieval-harvest-rule

IF the goal is to determine the number of workers

AND an instance has been retrieved with W workers

THEN set the number of workers to W

Given ACT-R's activation mechanism, the chunk that has been used or recreated most often and has the largest similarity to the current goal will be retrieved, for example

Transition1236

Isa sugar-goal

Previous-output 6

Workers 8

Output 9

Although this example does not exactly match the current goal, it is close enough if it has enough activation (it will be penalized for the mismatch between 6 and 7 in the previous-output slots). Based on this example the model will choose 8 as its next workforce. As the model gathers more experiences, its decisions will also improve, despite the fact that is does not have an explicit representation of the relationships between the task variables. In the experiment, participants improved their on-target decisions from 8 in the first block 40 trials to 15 in the second block of 40 trials. The model matched this fairly well with 9 and 14 trials, respectively.

3. Competing Strategies

Although instance learning is a powerful method to improve performance, there is no real evaluation of the knowledge used. This is an advantage for situations where no information for evaluation is available. If such information is available it can be inserted into the instance, but this does not translate into a higher activation. To directly influence knowledge parameters on the basis of an evaluation, we need to learn the utility of knowledge. An automatic process to keep track of utility is part of ACT-R's procedural memory. One way to use utility learning is to implement several problem-solving strategies using production rules, and have the mechanism keep track of the relative merits of these strategies.

With each production rule, ACT-R maintains two parameters: the estimated cost of the rule, and the estimated probability of success. The utility of a production i is defined as:

$$U_i = P_i G - C_i$$ **Production Utility Equation**

where P_i is an estimate of the probability that if production i is chosen the current goal will be achieved, G is the value assigned to that current goal, and C_i is an estimate of the cost (typically measured in time) to achieve that goal. As we will discuss, both P_i and C_i are learned from experience, while G is an architectural parameter.

The utilities associated with productions are noisy and on a cycle-to-cycle basis there is a random variation around the expected value given above. The highest-valued production is always selected but on some trials one might randomly be more highly valued than another. If there are n productions that currently match, the probability of selecting the ith production is related to the utilities U_i of the n production rules by the formula:

$$\mathrm{P}(i) = \frac{e^{U_i/t}}{\sum_j^n e^{U_j/t}}$$ **Production Choice Equation**

where the summation is over all applicable productions and t controls the noise in the utilities. Thus, at any point in time there is a distribution of probabilities across alternative productions reflecting their relative utilities. The value of t is about .5 in our simulations and this is emerging as a reasonable setting for this parameter.

Learning mechanisms adjust the costs C_i and probabilities P_i that underlie the utilities U_i according to a Bayesian framework. Because the example below is concerned with the learning of the probabilities, we will expand on that but the learning of costs is similar.

The estimated value of P is simply the ratio of successes to the sum of successes and failures:

$$P = \frac{Successes}{Successes + Failures}$$ **Probability of Success Equation**

However, there is a complication here that makes this like a Bayesian estimate. This complication concerns how the counts for Successes and Failures start out. It might seem natural to start them out at 0. However, this means that P is initially not defined and after the first experience the estimate of P will be extreme at either the value 1 or 0 depending on whether the first experience was a success or failure. Rather P is initially defined as having a prior value θ and this is achieved by setting Successes to θV + m and Failures to (1-θ)V + n where m is the number of experienced successes, n is the number of experienced failures, and V is the strength of the prior θ. As experience (m+n) accumulates P will shift from θ to m/(m+n) at a speed controlled by the value of V.

The value of the cost parameter C is estimated in a similar way: the sum of the efforts invested in a goal is divided by the total number of experiences (both successes and failures):

$$C = \frac{\sum_j effort_j}{successes + failures}$$ **Cost Equation**

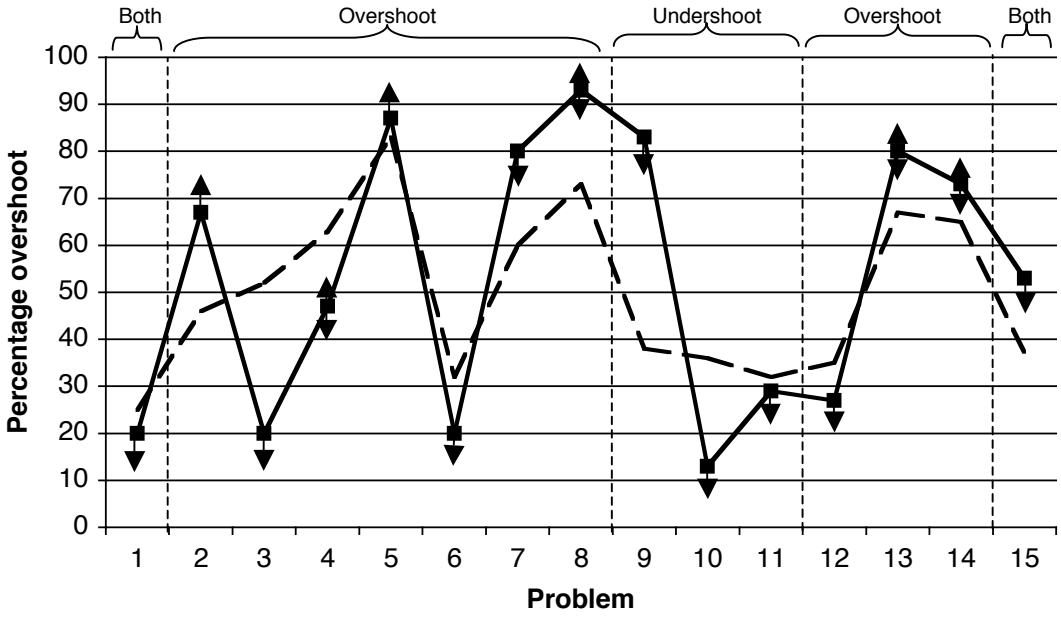
Utility learning is a useful mechanism in tasks where there are multiple cognitive strategies, but where it is unclear which of these strategies is best. The basic setup of a model using competing strategies is to have a set of production rules for each of the strategies. One of these production rules initiates the strategy, and this rule has to

compete with the rules that initiate the other strategies. As these rules gain experience, their parameters will reflect their utility, and ACT-R will tend to select the strategy with the highest utility. Depending on the level of utility noise the other strategies will also be sampled occasionally. This makes the system somewhat sensitive to changes in the utility of strategies.

Example model: The Building Sticks Task

In the Building Sticks Task (BST), participants have to construct a stick of a certain length using an unlimited supply of sticks of three other lengths (Lovett & Anderson, 1996). For example, the goal might be to build a stick of length 125 using sticks of length 15, 250 and 55. The goal can be reached by either addition or subtraction, so building a 125 stick can be achieved by 55+55+15 or by 250-55-55-15. Instead of being presented with the numbers, sticks of the appropriate length are shown on a computer screen, giving only an approximate idea of the real length of a stick. Participants started with a stick of length 0, and could subsequently select one of the three sticks to either add or subtract to or from the current stick. A consequence of only showing approximate lengths is that participants could not calculate the results of their actions beforehand, and had to build the goal stick by trial and error. The task was constructed in such a way that there was always one stick longer than the goal stick, while the other two are shorter, and that one of two strategies called undershoot and overshoot would lead to the goal. In the undershoot strategy, the two smaller sticks are used to achieve the goal, first by adding the larger of the two as many times as possible without exceeding the goal, and then continue with the smaller one. In the overshoot strategy the stick that is larger than the goal is selected first, and then the two smaller sticks are subtracted from that stick in a

manner similar to the undershoot strategy until the goal is reached. In the example above, both undershoot and overshoot lead to the goal. In almost all of the trials in the experiment, only one of the two led to the goal, while the other just missed it. For example the goal might be 101, and the building sticks 10, 155 and 22. The solution can be reached through overshoot (155-22-22-10), but not through undershoot.



Although there are only two strategies on the surface level, they can be chosen for different reasons. One can prefer to always use overshoot or undershoot, but another possibility is to let it depend on which initial step gets you closer to the goal, thereby following a hill-climbing heuristic. For example, in the case of the problem with 15-250-55 sticks and a goal of 125, the 55 stick brings one much closer to the goal than 250. In the problem with 155-22-10 sticks and a goal of 101, the 155 stick is closer to the goal. The solid line in Figure 2 shows the result of one of Lovett's experiments. In problem 1 and 15, both undershoot and overshoot will lead to the goal, in problems 2 to 8 and 12 to 14 only overshoot, and in problem 9 to 11 only undershoot. The data points indicate the percentage of participants that tried overshoot first. The arrows on the data points indicate

which strategy hill-climbing (i.e. the selection first of the stick closest to the goal stick) would favor: an arrow down means undershoot and an arrow up means overshoot. A double arrow indicates that there is no clear difference. The figure clearly shows that up to problem 6, participants tend to follow a hill-climbing strategy. For example on problem 5 the arrow points up, meaning overshoot is favored by hill-climbing, and over 80% of the participants select overshoot. By problem 7 they have discovered that up to then, the only strategy that works is overshoot, so they start using it all the time. However, by problem 9 overshoot is not longer successful, and participants adapt their strategy choice almost immediately. The same is true for problems 12 to 14, where overshoot is once more the successful strategy.

The basis for Lovett's model consists of four competing production rules:

1. Always choose overshoot
2. Always choose undershoot
3. Decide for overshoot when the large stick clearly brings you closer to the goal than the middle stick
4. Decide for undershoot when the middle stick clearly brings you closer to the goal than the large stick

Initially, rules 3 and 4 were given a slightly higher utility value than rules 1 and 2, indicating an initial preference for a hill-climbing strategy. However, once the model starts interacting with the experiment, it adjusts its utility values according to experience. The dotted line in the figure shows the models predictions. By problem 7 rule 4 (decide undershoot) has sufficiently dropped in utility to allow rule 1 (always overshoot) to win the competition and select overshoot despite the fact that undershoot brings you closer to

the goal. In a similar fashion the model adjusts its behavior according to the successes or failures of the four rules later in the experiment.

Individual differences

Cognitive models have the potential to go beyond modeling averages by having models that exhibit variability of behavior, or even fit models to individual participants. Individual differences can be explored at many levels, including knowledge and strategy variations, but up to now mainly the variation of global architectural parameters has been explored, more specifically working memory capacity. ACT-R has no separate working memory, but the effects of a limited capacity for unrelated elements can be simulated by decay and interference in declarative memory. Lovett, Reder and Lebiere (1999) found that individual differences in working memory capacity can be modeled by varying one ACT-R parameter: W, which controls the amount of spreading activation from the goal. To properly explain this, reconsider the activation equation:

$$A_i = B_i + \sum_j W_j S_{ji}$$ **Activation Equation**

As has been related earlier, the S_{ji} parameters represent the strengths of association between chunks. They are set to S-ln(fan_j) where fan_j is the number of chunks associated to chunk j. In many applications S is estimated to be about 2. The W_j's reflect the attentional weighting of the elements that are part of the current goal, and are set to W/n where n is the number of elements in the current goal, and W is a global ACT-R parameter that is by default set to 1. Lovett et al. (1999) explore variation of W in a model of the modified digit span (MODS) task. In this task participants had read aloud

sequences of characters made up of letters and digits. After the reading phase they had to recall the digits in the sequence. The number of digits that had to be recalled varied between 3 and 6. The characters were presented at a pace that made it very hard for the participants to do rehearsal. Figure 3 shows the performance curves of three of the participants and makes it clear that there are large individual differences. The model of the task is very simple: during the study phase the digits are stored in declarative memory. In the recall phase, the digits have to be retrieved from memory. The probability of success depends on the level of activation (see the Probability of Retrieval Equation). To model individuals, Lovett et al. varied the W parameter to match each of the individual performance profiles: a higher W corresponds with a higher activation, and therefore with a higher probability of recall. The figure shows three examples of this, fitting the data (the model is the dotted line) with values of W of 0.7, 1.0 and 1.1. The model matches not only the aggregate performance level but the detailed recall pattern as well.

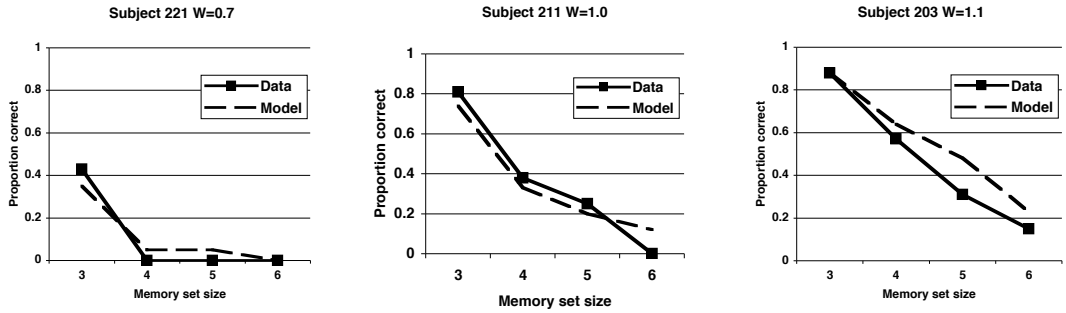


Figure 3. Results from three participants on the MODS task with model predictions, from Daily et al. (1999)

Apart from the W parameter there are other parameters in ACT-R that can account for individual differences. For example, Taatgen (2002) showed that W, the speed of production rule learning and the psycho-motor speed correlate with performance at different stages of the learning process.

4. Perceptual and motor processes

Each of the previous three modeling paradigms seeks to constrain the cognitive theory, either by learning or by capacity limitations. Another source of constraints is interaction with the outside world. As has already been shown in Figure 1, ACT-R has several modules that communicate with the outside world. These modules are adapted from the EPIC cognitive architecture developed by Meyer and Kieras (1997). The approach involves modeling, in approximate form, the basic timing behavior of the perceptual and motor systems, the output of the perceptual systems, and the input to the motor system.

An Example of Perceptual Modules in Parallel

The ACT-R model described by Byrne and Anderson (2001) for the Schumacher, et al. (1997; also reported in Schumacher et al. 2001) experiment is a useful illustration of how the perceptual-motor modules work together. It involves interleaving multiple perceptual-motor threads and has little cognition to complicate the exposition. The experiment itself is interesting because it is an instance of perfect time-sharing. It involved two simple choice reaction time tasks: 3-choice (low-middle-high) tone discrimination with a vocal response and 3-choice (left-middle-right) visual position discrimination with a manual response. Both of these tasks are simple and can be completed rapidly by experimental participants. Schumacher, et al. (1997) had participants train on these two tasks

separately, and they reached average response times of 445 ms for the tone discrimination task and 279 ms for the location discrimination task. Participants were then asked to do the two tasks together with simultaneous stimulus presentation and they were encouraged to overlap processing of the two stimuli. In the dual-task condition, they experienced virtually no dual-task interference—283 ms average response time for the visual-manual task and 456 ms average response time for the auditory-vocal task.

Byrne and Anderson (2001) constructed an ACT-R model of the two tasks and the dual-task that makes use of the perceptual and motor modules. A schedule chart for the dual-task model is presented in Figure 4. Consider the visual-motor task first. There is a quick 50 ms detection of the visual position (does not require object identification), a 50 ms production execution to request the action, followed by the preparation and execution of the motor action. With respect to the auditory-vocal task, there is first the detection of the tone (but this takes longer than detection of visual position), then a production executes requesting the speech, and then there is a longer but analogous process of executing the speech. According to the ACT-R model, there is nearly perfect time sharing between the two tasks because the demands on the central production system are offset in time.

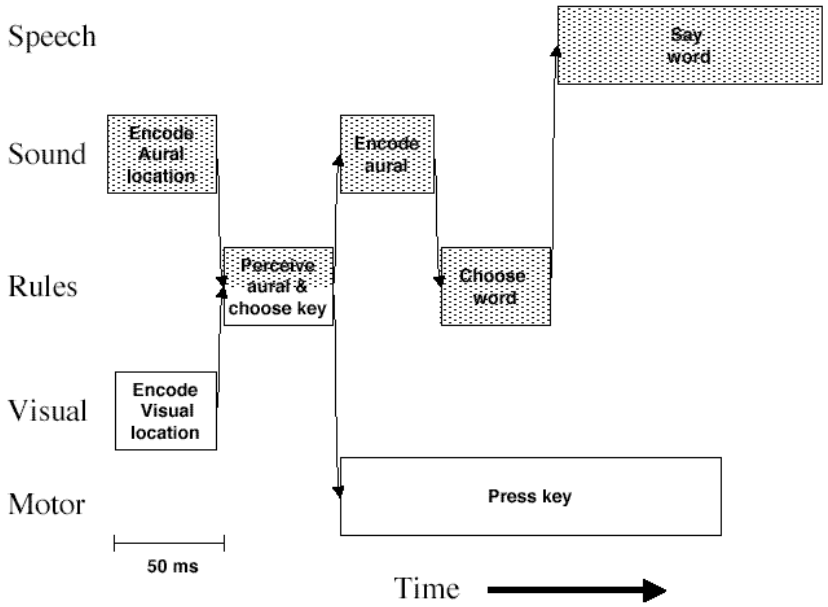


Figure 4. Timeline of how the model performs the Schumacher et al. task

5. Specialization of task-independent cognitive strategies

Production system models are often criticized for the fact that they have all the task-relevant strategies already encoded into their set of production rules, while in reality people first have to construct a representation of the task. For example, in the Byrne and Anderson model of perfect dual-tasking the relevant production rules have to be learned first (but see Anderson, Taatgen & Byrne, submitted, for a model that learns the rules). Indeed, participants can only achieve perfect dual-tasking after several days of training. Another goal of rule learning is to generalize from examples. In the sugar factory example discussed earlier generalization was based on single examples. In many other learning situations one can capitalize on regularities found in multiple examples. These

goals are achieved in ACT-R by a combination of production compilation, a mechanism that learns new rules, and general cognitive strategies.

Production compilation (Taatgen & Anderson, 2002) learns new rules by combining two existing rules that fire in sequence into one new rule. If the first of the two rules makes a request to declarative memory the result of which is used by the second rule, then the retrieved chunk is substituted into the new rule, effectively eliminating the retrieval. In itself this mechanism only produces more efficient and specialized representations of knowledge that is already available. When the production rules that are compiled are a general cognitive strategy, however, the resulting rules, although specializations of the general strategy, nevertheless generalize the specific experience. An example of this is learning the regular past tense rule in English (Taatgen & Anderson, 2002). A straight-forward strategy for finding past tense is to try to apply instance retrieval with a rule like:

Retrieve-past-tense

IF the goal is to find the past tense of a word word

THEN issue a request to declarative memory for the past tense of word

If this rule finds the past tense of the word, then a second rule uses this as the answer. The interesting case is when declarative memory does not produce the requested past tense, but (through partial matching), a different past tense. In that case we can apply an analogy strategy: find a pattern in the retrieved example and apply it to the current word. Suppose the retrieved example is a regular verb, then we can apply the pattern to our goal with a rule like:

Analogy-find-pattern

IF the goal is to find the past tense of word word1

AND the retrieval buffer contains past tense word2-suffix of word2

THEN set the answer to word1-past

Combining the two rules while substituting the retrieved word produces the regular rule:

Learned-rule

IF the goal is to find the past tense of a word word

THEN set the answer to word-ed

In this example the general cognitive strategy of analogy is compiled into a task-specific rule that generalizes a regular example. Other strategies that have been used in combination with production compilation are search for differences (van Rijn et al, 2003) and interpretation of instructions (Anderson et al, in press; Taatgen & Lee, 2003).

A more elaborate illustration of production compilation is based on a simplified air traffic control task (KA-ATC; Ackerman, 1988). The model of the task is explained in detail in Taatgen (2002) and Taatgen and Lee (2003). In this task, participants land planes by choosing a plane that is waiting to be landed and designating the runway on which the plane should land. There are four runways, the use of which is restricted by rules that relate to the length of the runway, the current weather, and the type of plane that is to be landed. For example, a DC10 can only be landed on a short runway if the runway is not icy and the wind is below 40 knots. Although participants receive an extended

instruction, they tend to forget some rules—especially the more complicated ones regarding weather, plane type, and runway length. The goal of the model is to capture the learning in this task by predicting the improvement in performance of the participants at both a global level and at the level of individual keystrokes.

An example of a production rule from the air traffic control task is:

Expert-ATC-rule
IF The goal is to land a plane and a plane has been selected that can be landed on the short runway (match of goal buffer)
AND you are currently looking at the short runway and it is not occupied (match of visual buffer)
AND the right hand is not used at this moment (match of manual buffer)
THEN note that we are moving to the short runway (change to goal buffer)
AND push the arrow-down key (change to manual-buffer)
AND move attention to the weather information (change to visual buffer)

This rule reflects the knowledge of an expert on the task at the stage in which a plane has been selected that has to be directed to the short runway. After checking whether the short runway is available, it issues the first motor command, and also initiates an attentional shift to check the weather, information regarding which might be needed for landing the next plane.

Although this example rule is very efficient, it is also highly task-specific; rules like this have to be learned in the process of acquiring the skill. For novices, the model assumes that all the task-specific knowledge needed about air traffic control is present in declarative memory, having been put there by the instructions given to participants. This knowledge has a low activation because it is new, and might have gaps in it in places where the participant did not properly memorize or understand the instructions. The production rules interpret these instructions and carry them out. Two examples of interpretive rules are:

Get-next-instruction-rule

IF the goal is to do a certain task and you have just done a certain step (goal buffer)

THEN request the instruction for the next step for this task (retrieval buffer)

Carry-out-a-push-key-rule

IF the goal is to do a certain task (goal buffer)

AND the instruction is to push a certain key (retrieval buffer)

AND the right hand is available (manual buffer)

THEN note that the instruction is carried out (goal buffer)

AND push the key (manual buffer)

A characteristic of interpreting instructions is that it results in behavior that is much slower than that of experts: Retrieving the instructions takes time, and during this time not much else happens. Also, parts of the instructions might be forgotten or misinterpreted, leading to even more time loss. In such cases, the model reverts to even more general strategies, such as retrieving past experiences from memory:

Decide-retrieve-memory-rule

IF you have to make a certain decision in the current goal (goal buffer)

THEN try to recall an experience that is similar to your current goal (retrieval buffer)

Decide-on-experience-rule

IF you have to make a certain decision in the current goal (goal buffer)

AND you have retrieved a similar experience that went well (retrieval buffer)

THEN make the same decision for the current goal (goal buffer)

This experience-based retrieval strategy retrieves the experience with the highest activation from declarative memory and is based on the assumption that experiences with a high activation are potentially the most relevant in the current situation.

The transition from novice to expert is modeled by production compilation. This mechanism again takes two existing rules that have been used in sequence and combines them into one rule, given that there are no buffer conflicts (for example, as would be the case when both rules specify using the right hand). For example, the two rules that retrieve an instruction and push a key, together with the instruction to press “enter” when the arrow points to the right plane during landing, would produce the following rule:

Learned-enter-rule

IF the goal is to land a plane and your arrow points to the right plane (goal buffer)

AND the right hand is available (manual buffer)

THEN note that the instruction is carried out (goal buffer)

AND push enter (manual buffer)

A rule that retrieves and uses old experiences can also be the source for production compilation. For example, in a situation in which the plane to be landed is a DC10 and the runway is dry, and a previous example in which such a landing was successful on the short runway is retrieved, the following rule would be produced:

Learned-DC10-rule

IF you have to decide on a runway and the plane is a DC10 and the runway is dry (goal buffer)

THEN decide to take the short runway (goal buffer)

New rules have to be recreated a number of times before they can compete with the parent rule, but once they are established they can be the source for even faster rules. Eventually the model will acquire a rule set that performs like an expert. Comparisons with data from experiments by Ackerman (1988; see Taatgen & Lee, 2003) show that the model predicts the overall performance increase (in terms of number of planes landed) and the individual sub-tasks (e.g., how much time is taken to land a single plane) very well. The model also does reasonably well at the level of individual keystrokes. As an illustration, Figure 5 shows the actual and predicted number of planes that are landed in a 10 minute trial, and the average time to land a plane.

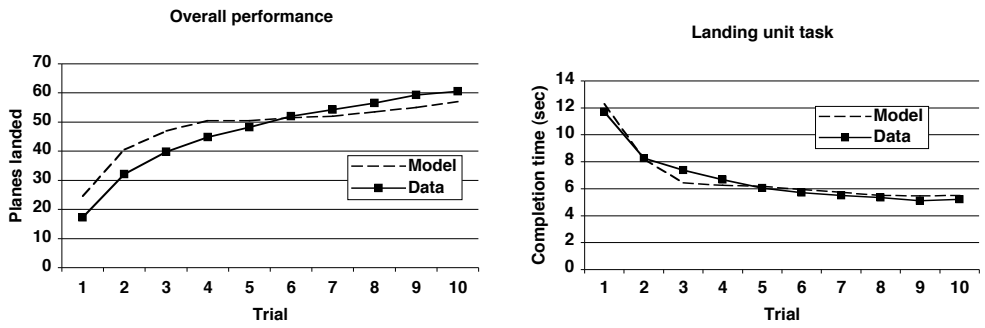


Figure 5. Numbers of planes landed and landing completion times for trials 1 to 10, data and model predictions

6. Which paradigm for what problem?

This chapter offers five modeling paradigms that are used frequently by ACT-R modelers. This enumeration is by no means exhaustive nor exclusive: there are models that use other methods, and there are models that use multiple modeling paradigms. The model of the Air Traffic Control task, for example, uses instance learning, perceptual and motor constraints, and has been used to explore individual differences. Nevertheless modeling paradigms can be guidelines and sources of inspiration for setting up a model for a new task. We will start by contrasting the three learning paradigms: instances, utilities and rules. Utility learning is a useful paradigm in tasks where the possible strategies are relatively clear, and where it can be assumed that people already have some sort of procedural representation of these strategies. When choosing this paradigm it is important to carefully think about how to define the set of strategies. For example, in the building sticks task it is not enough to just have undershoot and overshoot as strategies. To model the participants behavior, a third strategy, hill climbing, is necessary. Utility learning can also play a role in models where strategies emerge. In the model of the past

tense, for example, the learned regular rule eventually has a lower utility than retrieval of instances, explaining why it is only used if retrieval fails.

Instance learning is useful in situations where the underlying structure is unclear or absent. In alphabet arithmetic, there is no structure in the individual additions, so they have to be learned individually. In the sugar factory task there is an underlying rule, but it is unknown to the participants, and hard to derive on the basis of the behavior of the system. It is therefore suitable for tasks normally associated with implicit learning (Wallach & Lebiere, 2003). Instance learning allows for limited generalization, especially when used in combination with partial matching. In order for generalization to work though, it must be possible to retrieve the appropriate instance on the basis of activation. Such an instance can be found on the basis of similarity (instances similar to the current goal are useful), or frequency (instances that are retrieved or encountered often are useful). Instance learning cannot take utility into account directly, because a single production rule is responsible for the retrieval process. Instance learning is also less suitable for cases where extrapolation from the examples is needed: if the goal is too far outside the space spanned by the instances, the instance retrieved will probably not be very useful. In constructing an instance-based learning model it is important to carefully consider what should be stored in an instance. In some models a sequence of the last few actions is stored in an instance instead of just the last action (for example, in a model of sequence learning by Lebiere & Wallach, 2001). Production rule learning can unify the two approaches: rules can be learned out of the instance retrieval process, and these learned rules compete on the basis of utility. In order to learn interesting rules, some analogy-like process is necessary to use a retrieved instance that is not identical to the

goal. Production rule learning also enables learning from instructions, and therefore supplies the most accurate and complete form of modeling in experimental situations where a participant is supplied with instructions for a task and is asked to perform it. Models using this approach are more complicated though, so with any task it is worthwhile to investigate whether one of the more simple paradigms is sufficient. Modeling perceptual and motor processes also makes a model more complicated, but may substantially increase the precision of its predictions. Apart from providing precise predictions of the timing of perceptual and motor processes, the approach also acknowledges that cognition is not just a process in the head, but also an interaction with the outside world. For example, it might not be necessary to store information on the screen in the goal, as long as it is available in the visual buffer.

Although most ACT-R models on individual differences have focused on working memory capacity, it is nevertheless interesting to explore variability of behavior in a broader sense in cognitive models. When modeling behavior that is highly variable it is worthwhile to investigate whether noise in the model is sufficient to explain it (e.g. the AMBR model of Lebiere, Anderson & Bothell, 2001). Varying parameters like W can provide for additional variability. But individual differences in task performance can also be due to the fact that different individuals employ different strategies. Determining the precise content of an individual's knowledge and strategies is an arduous task, but programs have been proposed to meet it (Gobet & Ritter, 2000).

7. Summary

The focus of the example models in this chapter has been the modeling of human performance data, and not really on modeling agents in multi-agent systems. The main reason is that this is ACT-R's research focus, and that most modeling projects involve the development of cognitive models that produce predictions that are matched to human data. Nevertheless ACT-R can be used to program agents that exhibit human-like behavior, or serve as a theoretical basis to allow agents to construct a model of their user. The five modeling paradigms discussed can serve as basic tools or templates for such models. Examples of this can be found in chapter 4, where ACT-R is used to play two player games. In these games it is important to predict the actions of the opponent in order to anticipate them. Instance learning can be used to keep track of behavioral patterns in the opponents moves, enabling prediction of their most likely next move. Competing strategies may also be important in game playing: there may be a multiple strategies that can be brought to bear on a game, and their utilities may shift if the opponent also adjust their strategies.

Perceptual and motor constraints can be particularly important for agents immersed in virtual environments meant to recreate the sensory constraints of the real world, such as in first-person shooter games. Whether they are all used in a given model or not, all aspects of human cognition are important in some respect in producing human-like agents.

Acknowledgements

This research was supported by NASA grant NCC2-1226, ONR grant N00014-96-01491 and NWO/TOKEN/I2RP grant 634.000.002.

References

Ackerman, P. L. (1988). Determinants of individual differences during skill acquisition: Cognitive abilities and information processing. Journal of experimental psychology: General, 117(3) , 288-318.

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

Anderson, J.R., Bothell, D., Byrne, M.D., Douglass, S., Lebiere, C., & Qin, Y. (in press). An integrated theory of mind. Psychological Review. http://act-r.psy.cmu.edu/papers/403/IntegratedTheory.pdf

Anderson, J. R., Fincham, J. M. & Douglass, S. (1999). Practice and retention: A unifying analysis. Journal of Experimental Psychology: Learning, Memory, and Cognition, 25, 1120-1136

Anderson, J. R. & Schooler, L. J. (1991). Reflections of the environment in memory. Psychological Science, 2, 396-408.

Anderson, J. R., Taatgen, N. A. & Byrne, M. D. (in preparation). Learning to Achieve Perfect Time Sharing: Architectural Implications of Hazeltine, Teague, & Ivry (2002)

Berry, D. C., & Broadbent, D.E. (1984). On the relationship between task performance and associated verbalizable knowledge. The Quarterly Journal of Experimental Psychology, 36A, 209-231.

Byrne, M. D., & Anderson, J. R. (2001). Serial modules in parallel: The psychological refractory period and perfect time-sharing. Psychological Review, 108, 847-869.

Daily, L.Z., Lovett, M.C., & Reder. L.M. (1999). A computational model of individual differences in working memory capacity. Cognitive Science, 25, 315-353.

Gobet, F., & Ritter, F. E. (2000). Individual data analysis and Unified Theories of Cognition: A methodological proposal. Proceedings of the 3rd International Conference on Cognitive Modelling (pp. 150-157). Veenendaal, The Netherlands: Universal Press.

Lebiere, C, Anderson, J. R., & Bothell, D. (2001). Multi-tasking and cognitive workload in an ACT-R model of a simplified air traffic control task. Proceedings of the Tenth Conference on Computer Generated Forces and Behavior Representation. Norfolk, VA.

Lebiere, C. & Wallach, D. (2001). Sequence Learning in the ACT-R Cognitive Architecture: Empirical Analysis of a Hybrid Model. In R. Sun & C. L. Gilles (Eds.). Sequence Learning: Paradigms, Algorithms, and Applications (pp. 188-212). Berlin: Spinger Lecture Notes in Computer Science.

Logan, G. D. (1988). Toward an instance theory of automatization. Psychological Review, 95, 492-527.

Lovett, M. C., & Anderson, J. R. (1996). History of success and current context in problem solving: Combined influences on operator selection. Cognitive Psychology, 31, 168-217

Lovett, M. C., Reder, L. M., & Lebiere, C. (1999). Modeling working memory in a unified architecture: An ACT-R perspective. In: Miyake, A., & Shah, P. (Eds.),

Models of working memory: mechanisms of active maintenance and executive long-term memory activation, Cambridge University Press, New York.

Meyer, D. E. & Kieras, D. E. (1997). A computational theory of executive cognitive processes and multiple-task performance. Part 1. Basic mechanisms Psychological Review, 104, 2-65.

Schumacher, E. H, Seymour, T. L., Glass, J. M., Lauber, E. J., Kieras, D. E., & Meyer, D. E. (1997). Virtually perfect time sharing in dual-task performance. Paper presented at the 38th Annual Meeting of the Psychonomic Society, Philadelphia, PA.

Schumacher, E. H., Seymour, T. L., Glass, J. M., Fencsik, D. E., Lauber, E. J., Kieras, D. E., & Meyer, D. E. (2001). Virtually perfect time sharing in dual-task performance: Uncorking the central cognitive bottleneck. Psychological Science, 12 (2), 101-108.

Taatgen, N.A. (2002). A model of individual differences in skill acquisition in the Kanfer-Ackerman Air Traffic Control Task. Cognitive Systems Research, 3(1), 103-112.

Taatgen, N. A. & Anderson, J. R. (2002). Why do children learn to say "broke"? A model of learning the past tense without feedback, Cognition, 86 (2), 123-155.

Taatgen, N.A. & Lee, F.J. (2003). Production Compilation: A simple mechanism to model Complex Skill Acquisition. Human Factors, 45(1), 61-76.

Taatgen, N.A. & Wallach, D. (2002). Whether skill acquisition is rule or instance based is determined by the structure of the task. Cognitive Science Quarterly, 2(2), 163-204.

van Rijn, H., van Someren, M., & van der Maas, H. (2003). Modeling developmental transitions on the balance scale task. Cognitive Science, 27(2), 227-257.

Zbrodoff, N. J. (1995). Why is 9 + 7 harder than 2 + 3? Strength and interference as explanations of the problem-size effect. Memory & Cognition, 23 (6), 689-700.