| | |
|---|---|
| *author* | *N.A. Taatgen* |
| *title* | *A model of learning task-specific knowledge for a new task* |
| *status* | *submitted to (conference) Cognitive Science 1999* |

*February 1999*
*prepublication 1999-4*

# A model of learning task-specific knowledge for a new task

**Niels A. Taatgen**
Cognitive Science and Engineering, University of Groningen
Grote Kruisstraat 2/1, 9712 TS Groningen, the Netherlands
niels@tcw2.ppsw.rug.nl

**Abstract**

In this paper I will present a detailed ACT-R model of how the task-specific knowledge for a new, complex task is learned. The model is capable of acquiring its knowledge through experience, using a declarative representation that is gradually compiled into a procedural representation. The model exhibits several characteristics that concur with Fitt's theory of skill learning, and can be used to show that individual differences in working memory capacity initially have a large impact on performance, but that this impact diminished after sufficient experience. Some preliminary experimental data support these findings.

## Introduction

From the viewpoint of cognitive modeling it has always been hard to explain why people can learn new skills as fast as they usually do. In a typical psychological experiment, the participant is told to do something he or she has never done before. Nevertheless, only some verbal instructions and maybe one or two practice trials are enough to get them started. Initial performance is characterized by the fact that it is slow, and that many errors are made. Once a participant gains some practice, speed goes up, and the number of errors decreases. Fitt (1964) has described this process in three stages: a cognitive stage, in which processing is conscious, deliberate, slow, and requires full attention, an associative stage, in which processing gradually speeds up and less attention is needed, and finally an autonomous stage, in which a skill is performed very fast and requires very little attention. The autonomous stage is also characterized by the fact that deliberate control is gradually lost: it is hard for an expert in some domain to explain exactly what he does.

Generally, there are two categories of models of skill learning: production system models and neural network models. Models in the first category start out with a set of production rules that fully implement the skill. Several explanations for the speed-up produced by learning are offered. For example, in Soar (Newell, 1990) the speed-up is explained by chunking: reasoning steps that previously required multiple rules are carried out by a single rule after learning. Another explanation is that the efficiency of a rule itself is improved. For example, in ACT-R (Anderson & Lebiere, 1998) strength parameters are maintained for each rule. As a rule is practiced, its strength value increases, and the time it takes to use it decreases. Although these models often predict the data very well, a conceptual problem remains: where do these initial production rules come from? The general critique is that these models are "preprogrammed": they already contain the information they are supposed to learn. Also the more qualitative aspects of the stages in skill learning, like the requirement for attention at the start and the lack of conscious access in the end, remain largely unexplained.

The models in the second category are characterized by the fact that they acquire their knowledge only gradually. Neural networks are the most prominent examples of these models, although models based on genetic algorithms have the same type of properties. The big advantage of these models is that they are not "pre-programmed": they acquire the necessary knowledge autonomously. The disadvantage is that this process is slow. This means that the learning process towards to the autonomous stage is explained very well, but that learning within the cognitive stage is not captured.

One of the reasons why the initial stage of skill learning is so hard to model, is the fact that the participant's general common sense knowledge comes into play. This knowledge is necessary to interpret the instructions, and to fill in the gaps in these instructions. For example, if the instruction is "push the button when an X appears on the screen", it is assumed to participant knows what an X is, what the screen is, and how to push a button. In other cases, for example in the scheduling task I will discuss later on, participants have to discover for themselves how to perform the task.

A theory that can explain the transition from the cognitive to the autonomous stage is proposed by Anderson (1982). This theory is based on the distinction between declarative and procedural memory. Declarative memory contains factual knowledge, and is available to conscious access. Procedural memory on the other hand contains production rules. These rules can only act, and cannot be inspected themselves. The idea is that in the cognitive stage the task-specific knowledge is represented declaratively. Declarative knowledge cannot act by itself, so it has to be interpreted by production rules. This explains why processing in the cognitive stage is slow. It also explains the fact that it is a conscious process, since declarative knowledge is available to consciousness. Gradually, during the associative stage, this declarative knowledge is compiled in production rules. These rules can act much faster than declarative knowledge, but are not available to consciousness. When all declarative knowledge is compiled, the autonomous stage is reached. Since the declarative knowledge is no longer needed, it is gradually forgotten and conscious control of task performance is lost.

Although this theory of skill learning is specified in terms that can be implemented in a production system, this is hardly ever done. In this paper I will discuss a model that does acquire its skills in this fashion. The model is implemented in ACT-R, and the task it models is scheduling.

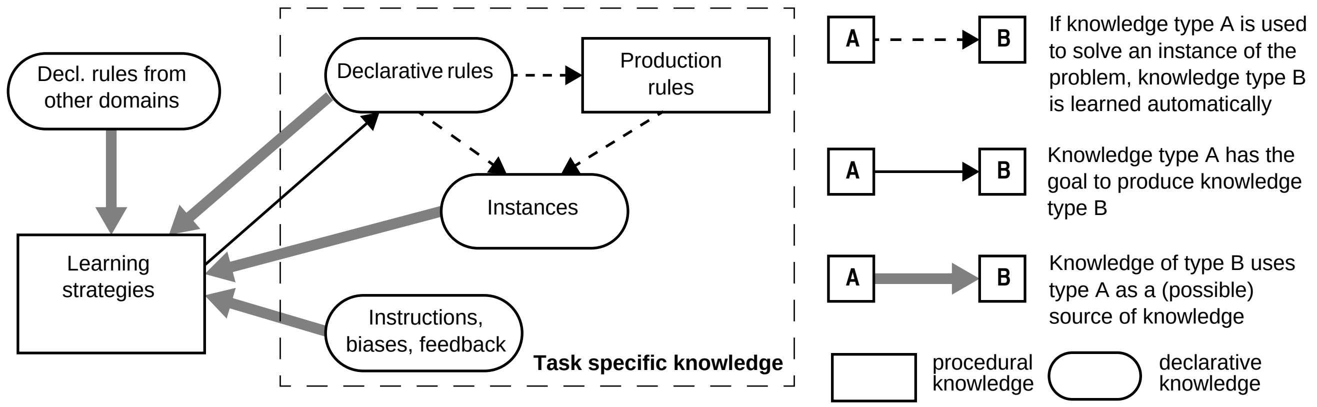

Decl. rules from other domains
Declarative rules
Production rules
Instances
Learning strategies
Instructions, biases, feedback
Task specific knowledge
A
B
If knowledge type A is used to solve an instance of the problem, knowledge type B is learned automatically
Knowledge type A has the goal to produce knowledge type B
Knowledge of type B uses type A as a (possible) source of knowledge
procedural knowledge
declarative knowledge

Figure 1: Overview of the proposed skill-learning model in ACT-R

## The general skill-learning model

ACT-R (Anderson & Lebiere, 1998) is a cognitive architecture based on a production system. It has two long-term memory stores: a declarative memory and a procedural memory. Although knowledge is represented by symbols in each of these memories (chunks and productions, respectively), it has a rich underlying sub-symbolic layer of representation that handles aspects like choice, errors, reaction times and forgetting.

Figure 1 shows an outline of the general skill-learning model. Each of the boxes in the diagram represents a type of knowledge: rectangles represent procedural knowledge, rounded boxes represent declarative knowledge. The dashed boundary indicates which of these knowledge types are task specific. The four types of knowledge that are part of the task-specific knowledge each have a different representation. Declarative rules are rules that are stored as a fact. An example of such a rule is:

Example-Declarative-Rule
  Isa Declarative-Rule
  Goal Count
  Retrieve Number-order
  Test "*current count is equal to first number in number-order*"
  Action "*set the count to the second number in the number-order*"

This rule is part of a counting procedure. It specifies that it is applicable to a goal of type count. When it is applied, a number-order fact (e.g., "Two is followed by Three") has to be retrieved of which the first number matches the current count. The second number in this fact should be stored in the goal. The procedural counterpart of this rule is:

IF the goal is to count and the current count is *num1* AND *num1* is followed by *num2*
THEN set the current count to *num2*

Each time a declarative rule is used, there is a small probability that it will be compiled into a production rule. Although both representations will lead to the same results, there are

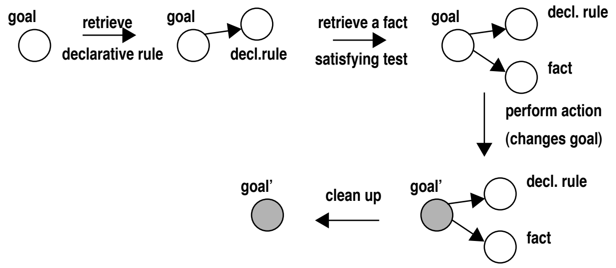
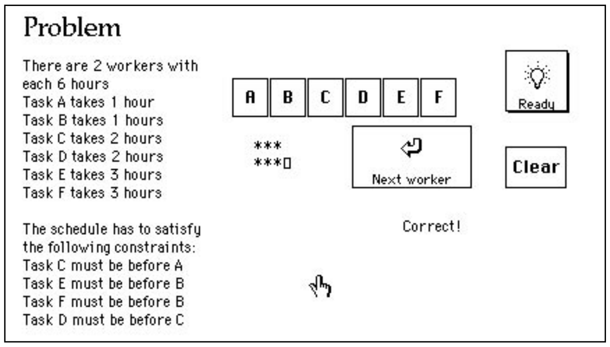


Figure 2: Interpreting declarative rules

differences. The declarative rule cannot act by itself. It needs to be interpreted by production rules. Figure 2 shows the process: given a certain goal, a suitable declarative rule is retrieved first, than the fact specified in the rule is retrieved while checking whether the test is satisfied at the same time. Finally the action is carried out and the goal is cleaned up. The procedural representation can do this in a single step, so is much faster. A declarative rule has its own advantages, however. Since it is declarative, it can be inspected by other rules than the rules that are used to carry it out. The counting rule, for example, can be modified slightly to count letters in stead of digits. Production rules do not offer this flexibility, since they cannot be inspected themselves. If a declarative rule leads to an error, a modified version can be created to try something else. A final aspect of using a declarative rule is that it uses working memory capacity. There is no such thing as a working memory in ACT-R. The function normally attributed to working memory, keeping track of currently relevant task knowledge, is related to the spread of activation from the goal. Due to the interpretation process of the declarative rule, the activation that originates from the goal has a larger fan: it is spread out over more chunks and becomes "thinner". In terms of working memory capacity: the declarative rule consumes some of the working memory capacity that is available for normal processing.

In stead of using a rule to solve a problem, an example or instance can be retrieved that immediately contains the answer, in a fashion that is comparable to Logan's (1988) instance theory. In ACT-R, achieved goals are kept in declarative memory automatically, an serve as instances that can be retrieved later (provided they have not been forgotten).



Problem

There are 2 workers with each 6 hours
Task A takes 1 hour
Task B takes 1 hours
Task C takes 2 hours
Task D takes 2 hours
Task E takes 3 hours
Task F takes 3 hours

The schedule has to satisfy the following constraints:
Task C must be before A
Task E must be before B
Task F must be before B
Task D must be before C

Correct!

Figure 3: Example of a scheduling experiment

Since there is no initial task-specific knowledge, except for some uninterpreted instructions and possible biases, general “common sense” knowledge is needed to make a start. This knowledge is indicated in figure 1 by the term *learning strategies*. Learning strategies interpret instructions or try to modify declarative rules for other, similar tasks, or use other strategies to come up with methods to do the new task. The general idea is that this set of learning strategies is not fixed, but may be a source of individual differences. Taatgen (1997), for example, describes how different learning strategies can explain the difference in behavior between adults and children in the discrimination-shift task. In the example model analogy will be used as a learning strategy.

At each stage in the reasoning process, there are multiple possible strategies to try. According to ACT-R, the strategy with the highest expected gain will win the competition. Retrieving an instance or using a production rule is generally the fastest strategy, followed by using a declarative rule. Using a learning strategy is generally only a good idea if the existing knowledge is insufficient or incorrect.

## The scheduling task

An example of a complex task that is easy to explain but which is new to most people is scheduling. Figure 3 shows an example of a scheduling task used in our experiments. The goal is to assign a number of tasks (6 in the example) to a number of workers (2 in the example), satisfying a number of order constraints. A solution to the example in figure 3 is to assign DEA to the first worker, and FCB to the second. The participants have to solve the problem entirely by heard: the interface only allows them to type in the answer, which is represented by asterisks on the screen. An example fragment of verbal protocol is as follows:

> Yes. There are two workers with each six hours. Two. Task A, task B, task C. The schedule has to satisfy the following constraints... Task C before A, C before A, E before B, F before B and D before C. [..unintelligible..] First now D. D.. D..C..A..B.., D..C..A..B.., D.C.A.B., DCAB, and then, DCAB, [keys in DCAB] and then E... E..F, E..F. [keys in EF] [Receives feedback] Oh, task F is not before B. C.., D has to be before C. D.. No, C..D.., D has to be before C. C.. D.., C.. D.., A...B [keys in CDAB] that’s one worker. E..F..., [keys in EF]. [receives feedback] Huh?! Task F is not before B and task D is not before C? Oh wait. D has to be before C, so first D... D...C..AB..AB [keys in DCAB]. Next worker, F.. yes, F..E.., ready. [keys in FE]. [receives feedback] Task E is not before B? Isn’t it? Yes? [Emphasizing, keys in] D..C..A..B..E..E..F...ready. [receives feedback]. Well! Ehmm.. Task D takes two hours. [Silence] Task F is not before B, so F should be before B. Task E before... E should be before B, so E and F shouldn’t be done by.... by the same worker. So we will, let’s see. Task C before A, so we will first.... E before B, so we will first E..E..E..B..C. E...E..B..C.., EBC, no that’s not right. EBC..F..A..B.. Ah.. start again. The D should be before C. [silence]. E... Ehm... The D should be before the C, so we put the D with worker one, and C with worker two. So we start with E with worker one... E..C..A.. E.C.A. ECA.. E.C.A. No, I don’t get it... E..C..A..D..F.. Oh.. wrong again.

As can be seen is this small fragment of protocol, the participant has a hard time memorizing partial solutions and deciding what to do next. This often leads to forgetting or mixing up letters, signs of an overloaded working memory.

I will explore this task in two models, the second of which is an extension of the first. The first will only model the proceduralization aspect of the general model from figure 1, while the second model will use a learning strategy to produce new task-specific declarative rules. The second model will also suffer from working memory limitations, as will be evident when I discuss its performance.

## The first model

A first approximation of a model of scheduling has the following components:

1. Production rules that interpret and proceduralize declarative rules as outlined in the previous section
2. A top-goal that reads the task and pushes a task subgoal and types out the answer of the subgoal has been reached
3. Productions that store elements in a list, and implement rehearsal, both maintenance and elaborate.
4. A set of declarative rules that implements a simple strategy for scheduling.
5. Productions that produce some sort of verbal protocol.

### Results of the first model

The model was tested using a set of ten example problems, all of which consisted of two workers and six or seven tasks. Although the problems are not particularly hard, this is not yet important since the answer given by the model is not checked. The model uses only symbolic learning, and has all subsymbolic learning turned off. New chunks in declarative memory do not have a role in the problem solving process yet. Improvements in performance can therefore be attributed to production compilation. Figure 4 shows the learning

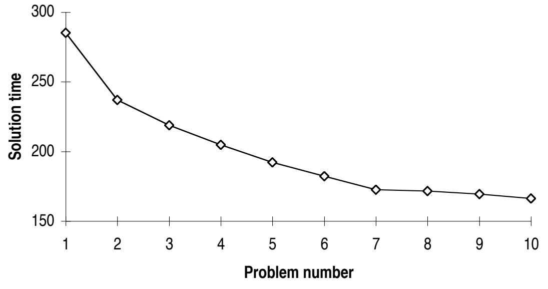


Figure 4: Learning curve of the first model

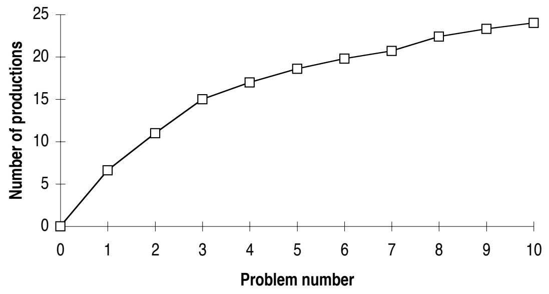


Figure 5: Growth in the number of production rules

curve of the model. To get some idea of the rate of learning, the growth in the number of productions is plotted in figure 5. The more interesting part is the pseudo verbal protocol produced by the model. To see the impact of proceduralization, examples of the output of the first and the tenth problem have been printed in below.

### Protocol of the first problem

There are two workers. Each of the workers has seven hours. Task A takes two hours. Task B takes two hours. Task C takes two hours. Task D takes two hours. Task E takes three hours. Task F takes three hours. Task B before F. Task F before A. First I will find a task to begin with. Let's look at an order constraint. B before F. Let's see if there is no earlier task. There is no earlier task. Begin with B. B Can we find a next task just by looking at the order? B before F. B.. F.. Can we find a next task just by looking at the order? F before A. B.. F.. A.. Can we find a next task just by looking at the order? Is this a schedule for one worker or for more? Now I am going to count how many hours we already have B.. How long does this one take? Task B takes two hours. Add this to what we have. nothing plus two equals two. F.. How long does this one take? Task F takes three hours. Add this to what we have. Two plus three equals five. A.. How long does this one take? Task A takes two hours. Do we have enough for one worker? Each worker has seven hours. We can move to the next worker.. B.. F.. A.. next.. Let's do the rest Now we are going to look at all the tasks, and see which ones are not yet in the schedule. Let's start with A. Task A takes two hours. Let's try to put it in the schedule. A is already in the schedule. OK, what is the next letter? B comes after A. Task B takes two hours. Let's try to put it in the schedule. B comes after A. B is already in the schedule. OK, what is the next letter? C comes after B. Task C takes two hours. Let's try to put it in the schedule. C comes after B. B.. F.. A.. next.. C.. OK, what is the next letter? D comes after C. Task D takes two hours. Let's try to put it in the schedule. D comes after C. B.. F.. A.. next.. C.. D.. OK, what is the next letter? E comes after D. Task E takes three hours. Let's try to put it in the schedule. E comes after D. B.. F.. A.. next.. C.. D.. E. Task F takes three hours. Let's try to put it in the schedule. F comes after E. F is already in the schedule. OK, that was the last task, we're done! The answer is B F A next C D E

### Protocol of the tenth problem

There are two workers. Each of the workers has six hours. Task A takes one hours. Task B takes one hours. Task C takes two hours. Task D takes two hours. Task E takes three hours. Task F takes three hours. Task D before E. Task E before A. First I will find a task to begin with. Let's see if there is no earlier task. Begin with D. D.. D.. E.. D.. E.. A.. Can we find a next task just by looking at the order? Is this a schedule for one worker or for more? Now I am going to count how many hours we already have D..E..A.. D..E..A.. next Now we are going to look at all the tasks, and see which ones are not yet in the schedule. Let's start with A. A is already in the schedule. D..E..A.. next.. B D..E..A..next..B..C D is already in the schedule. E is already in the schedule. D..E..A.. next.. B..C..F. OK, that was the last task, we're done! The answer is D E A next B C F

Clearly, the protocol of the first problem is a protocol analyst's dream, because participants are hardly ever that precise. But the tenth protocol looks more familiar: many steps in the process are omitted, and we can only guess why some decisions have been made. This concurs with the general idea that proceduralized skills produce no verbal protocol (Ericsson & Simon, 1984).

Although this first model shows some interesting properties similar to real problem-solving behavior, it is far from complete. The current model just takes a single shot at the solution, and does not retry if it is incorrect. Only ACT-R's production compilation had been turned on, so the model will never forget any intermediate results it has found. And finally, the model starts out with a set of task-specific declarative rules. One of the desired properties of the model was to start without any task-specific knowledge. These issues will be addressed in the second version of the model.

## Learning scheduling

People may not know anything about schedules, but they do know something about lists, and how to construct them. Suppose we need to make a schedule. We may use knowledge about lists to start with. How do we make a list? First we have to find a first item for the list, a begin. Once we have a begin, we find a next task until we are done. But how do we find something to begin with, and how do find a next task? We may choose to handle these problems making them subgoals, or we may try to find mappings between 'begin' and 'next' and terms in the scheduling problem. For example, a mapping can be made between 'next' and an order-constraint in the scheduling problem. The result is a modified version of the list-building declarative rules, with 'list' substituted by 'schedule' and 'next' substituted by 'order'. Note that for sake of the explanation, the terms 'list', 'begin', 'before' and 'next' will be used to refer to general terms, and 'schedule' and 'order' to refer to task-specific terms. Except for knowledge on how to build a list, the analogy between a schedule and a list may also offer knowledge on how to retain a list in memory by rehearsal.

Although these new declarative rules may find a start, they are insufficient to build a complete schedule, mainly because the mapping between 'next' and 'order' is inadequate. When this declarative rule fails to make a complete schedule, another plan may take over and contribute to the schedule.

An idea that may take over if the list-building plan fails to add any more tasks to the schedule is the plan that tries to complete the first worker. A useful general plan may state that whenever something has to be completed, the difference between the desired size and the current size has to be calculated, after which an object had to found with a size equal to this difference.

The central emerging idea is therefore that several strategies from similar domains are adopted and patched together. This method of adapting old strategies for new purposes is similar to *script* and *schema* theories. Traditional script and schema theories assume that a complete script is first adapted to fit the current task, and then carried out. The ACT-R model uses a more on-demand style of adaptation: a new declarative rule is created at the moment it is needed.

## The second model

The second model solves some of the shortcomings of the first. It learns new declarative rules as outlined in the previous section. Furthermore, the following aspects have been added to the model:

1. The model has to come up with a correct solution within 300 seconds. If no solution has been found in this period, the attempt is counted as a failure.
2. ACT-R's activation learning is turned on. As a consequence, the model can forget all kinds of partial results it derives, most notably the list that contains the partial solution, but also read constraints (which have to be reread in that case), newly derived declarative rules, etc.
3. Several extra declarative rules have been added to ensure that correct solutions are eventually found by the model. The model now tries to satisfy the order constraints for the second worker as well, and uses the feedback it gets when it makes an error as a starting constraint for the next try.

### Results of the second model

The following protocol fragment, produced by the model, gives an impression of the additional aspects of the model:

> There are two workers. Each of the workers has six hours. Task A takes one hours. Task B takes one hours. Task C takes two hours. Task D takes two hours. Task E takes three hours. Task F takes three hours. Task B before C. Task F before A. I have to think of some new way to find a schedule. Let's use what I know about lists. First I will find something to begin with. Let's look at a before constraint. I have to think of some new way to find a before. Let's use what I know about order. Let's use a order fact as a before fact. F before A. I have to think of some new way to find a before following a failed declarative rule 12. F before A. I have to think of some new way to find a before following declarative rule 12. Let's look at a before constraint. Let's see if there is no earlier element. Let's use a order fact as a before fact. There is no earlier element.
>
> [three failed episodes removed]
>
> First I will find something to begin with. Begin with F. F.. Now I have to find the next thing. F before A. A.. Now I have to find the next thing. No more items for the list, let's check whether we're done. F.. A.. Is this a schedule for one worker or for more? Now I am going to count how many hours we already have F.. Add this to what we have. Nothing plus three equals three. A.. Add this to what we have. Three plus one equals four. Do we have enough for one worker? No, the schedule is not full, yet. F.. A.. Now find the task that fits in. Task C takes two hours. C.. We can move to the next worker.. NEXT-WORKER Let's do the rest F.. A.. C.. NEXT-WORKER.. I now try to find any unused order constraints. B before C. B before C. This one hasn't been used, so the constraint has been found. B before C. B before C. B.. Now we are going to look at all the tasks, and see which ones are not yet in the schedule. Let's start with A. Task A takes one hours. A is already in the schedule. OK, what is the next task? Task B takes one hours. B is already in the schedule. Let's move on to the next task. OK, what is the next task? Task C takes two hours. C is already in the schedule. Let's move on to the next task. OK, what is the next task? Task D takes two hours. D.. Let's move on to the next task. OK, what is the next task? Task E takes three hours. E.. Let's move on to the next task. OK, what is the next task? Task F takes three hours. F is already in the schedule. Let's move on to the next task. OK, what is the next task? OK, that was the last task, we're done! F.. A.. C.. NEXT-WORKER.. B.. D.. E.. The answer is  F A C NEXT-WORKER B D E

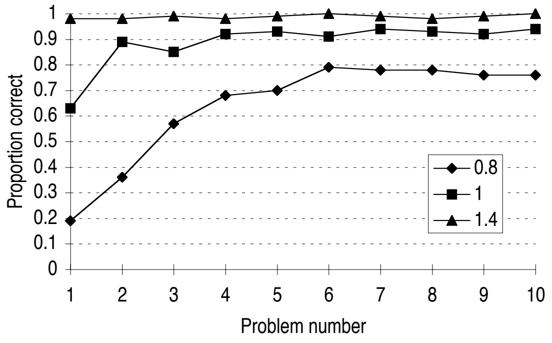


Figure 6: Accuracies of the second model for a low (0.8), average (1) and high working memory (1.4) capacity

This particular problem required five attempts, only two of which are shown: the first and the final, successful episode. In the first two fragments, the model is busy figuring out how aspects of the problem can be mapped onto things it knows something about. Unfortunately, the primitive protocol generating part of the model produces some awkward sentences with references to internal symbols. Somewhere along the line the model gets stuck, because it cannot keep track of all the constraints in the task and all the newly derived declarative rules. The third search episode is slightly more successful: it can use the declarative rules derived in the first two episodes. Unfortunately, it fails just when it is done, because it cannot retrieve the start of the list anymore for typing in the answer. In the fifth, successful episode some of the earlier derived results can be retrieved, which is evidence that the model uses the instance strategy. For example, the model immediately starts with "Begin with F" instead of deriving this fact.

### Individual differences

The basic performance results of the second model have roughly the same shape as the results from the first model. The second model is, however, sensitive to working memory limitations. According to Lovett, Reder and Lebiere (1997), individual differences in working memory capacity can be modeled by variations in the spread of activation in ACT-R, as mediated by the source activation (W) parameter. When this parameter is varied between 0.8 and 1.4, it produces the accuracy rations depicted in figure 6. Accuracy means in this case: the proportion of problems solved correctly within 300 seconds. The interesting aspect of different working memory capacities is that at the start of the experiment the differences are very high: the low capacity model hardly gets any problem done, and the high capacity model is successful almost all the time. At the tenth problem however, these differences have diminished: the low capacity model has proceduralized

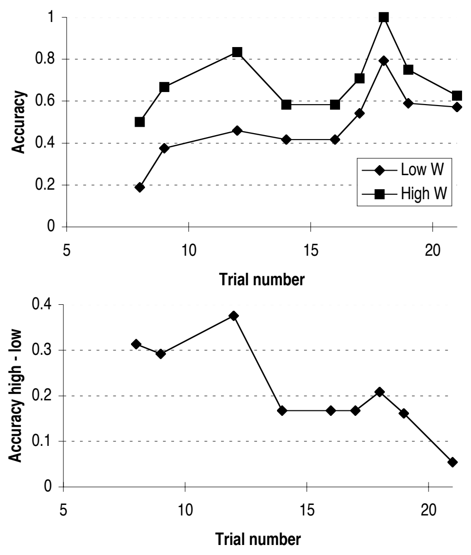


Figure 7: Proportion of correctly solved schedules for the 8 highest source activations and the 8 lowest source activations. The top graph shows the accuracy curves for both groups, the bottom graph shows the difference between the two.

enough knowledge to be able to reach a correct solution most of the time.

## Preliminary empirical evidence for the model

As part of a study of mental fatigue (Jongman & Taatgen, submitted), we have used the scheduling task and the digit working memory task that has been modeled in ACT-R by Lovett, Reder and Lebiere (1997). The digit working memory task was used to make an estimate of the working memory capacity of a participant, expressed in the ACT-R source activation parameter. This working memory capacity was related to the performance on the scheduling task. Unfortunately, the scheduling task as it was used in this particular experiment was a mixture of problems with two and three workers with varying difficulty and varying time limitations. It is therefore hard to directly compare the results to the model predictions. Nevertheless some of the more qualitative predictions of the model can be tested with respect to individual differences. The model predicts a strong correlation between working memory capacity and the performance on the scheduling task. This proved to be the case in the experiment: the correlation between the estimated source activation and the number of successfully solved schedules is 0.56 (with n=16). This correlation increases to 0.66 if the analysis is restricted to the three-worker schedules, the schedules that require most working memory capacity. A more specific prediction of the model is that the effect of working memory capacity on performance will diminish due to proceduralization. To investigate this prediction, the group of participants is split into eight low source-activation participants (W<0.95) and eight high source-activation participants (W>0.95). The proportions of correct solutions for each of the groups is plotted in figure 7. In this graph only three-workers problems are shown, and to average out part of the noise each data point is averaged with its predecessor and its successor. There is a clear convergence between the two curves, as can be seen in the bottom graph that depicts the difference.

## Discussion

The skill-learning model discussed above exhibits many of the characteristics normally attributed to different stages in problem solving. In the cognitive stage, declarative rules are used. While these rules are interpreted, they are part of the current goal context and are available to consciousness. The interpretation process is slow, and susceptible to errors, especially if the task-demands with respect to working memory are high. On the other hand declarative rules may be manipulated by learning strategies, offering flexibility. In the autonomous stage, all processing is handled by production rules, which are fast and less demanding for working memory. Moreover, once the declarative rule itself is forgotten, conscious access to the skill is lost.

Although the model discussed above implements the skill-learning process for the scheduling task, the basic framework can be used for learning other tasks as well. The prior knowledge of the model can to be changed, and maybe also the learning strategies, in order to enable it to learn a different task.

## Availability of models

Both models can be retrieved from: http://tcw2.ppsw.rug.nl/~niels/thesis. They are listed under the “chapter 7” caption.

## References

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

Anderson, J. R. (1982). Acquisition of cognitive skill. *Psychological Review, 89*, 369-406.

Ericsson, K. A. & Simon, H. A. (1984). *Protocol analysis. Verbal reports as data.* Cambridge, MA: The MIT Press.

Fitts, P. M. (1964). Perceptual-motor skill learning. In A. W. Melton (Eds.), *Categories of human learning*. New York: Academic Press.

Jongman, L. & Taatgen, N.A. (submitted). An ACT-R model of individual differences in changes in adaptivity due to mental fatigue. Submitted to the 21th cognitive science conference.

Logan, G. D. (1988). Toward an instance theory of automization. *Psychological Review, 22*, 1-35.

Lovett, M. C., Reder, L. M., & Lebiere, C. (1997). Modeling individual differences in a digit working memory task. *Proceedings of the Nineteenth Annual Conference of the Cognitive Science Society* (pp. 460-465). Hillsdale, NJ: Erlbaum.

Newell, A. (1990). *Unified theories of cognition*. Cambridge, MA: Harvard university press.

Taatgen, N.A. (1997). A rational analysis of alternating search and reflection in problem solving. *Proceedings of the 19th Annual Conference of the Cognitive Science Society* (pp.727-732). Hillsdale, NJ: Erlbaum