# How Cognitive Models can Inform the Design of Instructions

**Niels A. Taatgen (taatgen@cmu.edu), David Huss (dhuss@andrew.cmu.edu)**
**and John R. Anderson (ja@cmu.edu)**
Carnegie Mellon University, Department of Psychology, 5000 Forbes Av.
Pittsburgh PA, 15213, USA

**Abstract**

Instructions represented as lists of steps lead to inflexible and brittle behavior in cognitive models, suggesting that list-style instructions lead to poor learning in people as well. On the basis of this assumption we designed an alternative operator-style instruction that produces better learning in models. In an experiment and model of interacting with a simulated Flight Management System, a system that is notoriously hard to learn on the basis of list-style instructions, we show that alternative instructions produce significantly better and more robust learning.

## Introduction

Most cognitive architectures incorporate learning mechanisms, allowing them to learn new knowledge in the same way that humans learn new knowledge. Assuming the theories of learning the architectures propose are reasonably accurate, they can be applied to instruction: given a particular set of instructions, architectures should be able to predict how much practice it takes to reach expert behavior, what the quality of expert behavior is, and how well the skill generalizes to other situations.

We started our work on learning from instructions with models that assumed that instructions were memorized as lists of steps that had to be carried out, and that were organized in a hierarchical goal structure (Taatgen & Lee, 2003; Anderson et al., 2004). This assumption was based on many methods of task analysis that employ such a structure (for example GOMS, Card, Moran and Newell, 1983). Although these models managed to match the learning characteristics of the participants fairly well, they did have problems in getting the details right.

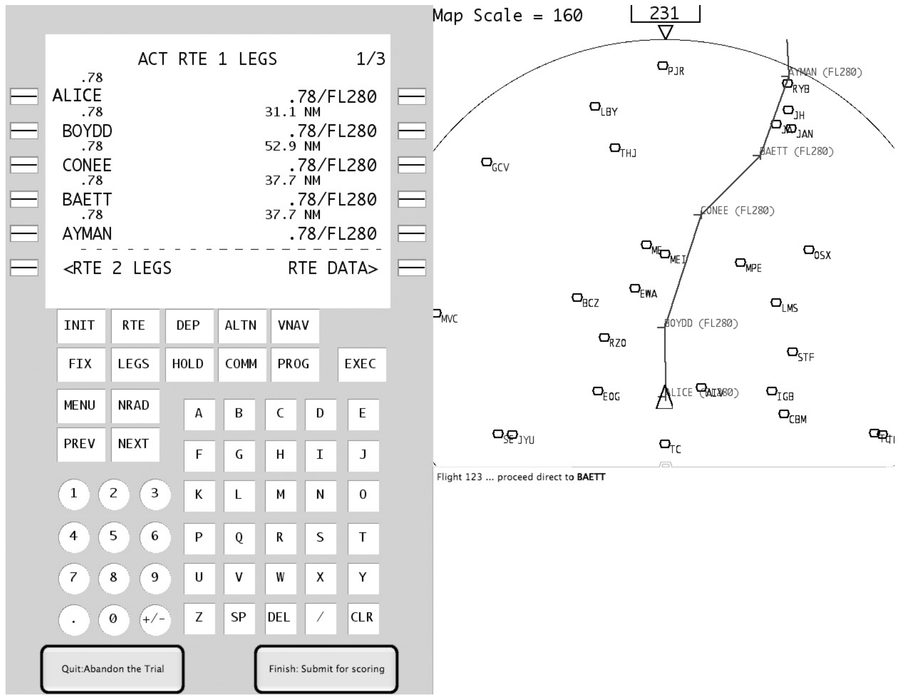
For example, in the Kanfer-Ackerman Air Traffic Controller task (KA-ATC, Ackerman, 1988) participants had to land airplanes on runways while observing certain constraints related to the weather. In order to land a plane, a plane-runway combination had to be found that satisfied all the constraints, after which a series of key presses had to be executed to enter this information into the system. The model we constructed of the task (Taatgen & Lee, 2003) matched both the global learning and the learning of the individual unit tasks very well, but consistently mispredicted the timing of the individual keystrokes. Consistent with the task analysis, the model would determine the runway-plane combination and then execute the keystroke series. However, the data showed a completely different pattern: the time before the first keystroke was so short that it was impossible for participants to already have finished the planning. Instead they were parallelizing planning and execution, something that was impossible to account for given the way that we represented instructions in the model.

There are two possible solutions to explain the apparent flexibility that humans show in parallelizing parts of a task. A first solution is to add more control to the model and design a process that overviews the scheduling of the various steps: a *top-down* controller that determines when a particular step can be done. We tried this solution for the KA-ATC task with mixed success (Lee & Taatgen, 2002). It is also the type of solution put forward by EPIC, where specific control knowledge is used to interleave multiple tasks (Kieras, Meyer, Ballas & Lauber, 2000). A second solution is to not increase the amount of control, but rather to decrease it and let the environment or current problem state help determine the next step, basically a *bottom-up* control strategy (Taatgen, 2005; Taatgen, in press). In that solution instructions are not organized in a tight hierarchy, but instead form a loose collection of operators that have conditions attached to them that specify when they can be carried out.

An example of a top-down solution in the KA-ATC task would be a look-ahead process that scans the task hierarchy for future motor tasks that can already be carried out during the planning stage of landing a plane. A bottom-up strategy on the other hand wouldn't employ a task hierarchy in the first place, but would rather have a set of instructions keyed to particular conditions, and would carry out instructions as soon as their conditions are satisfied. Once the planning instructions for the task have selected a particular plane to land, even if it is not the final choice, motor processes already start moving the cursor to the desired plane.

Although the top-down solution can work very well in an architecture like EPIC where central processes can be executed in parallel, it is problematic for ACT-R, because central processing is serial. The look-ahead processes needed and the book-keeping of which steps are already done consume so much extra time that this only makes performance worse instead of better. The bottom-up solution is much more attractive because it leads to more simple and more flexible models (Taatgen, 2005; Taatgen, in press). However, in order for any solution to work, the model needs more knowledge than a linear list of instructions, otherwise it cannot determine whether a future step in the list can already be carried out ahead of time (this would of course also be true for a top-down solution).

In order to illustrate the need for an augmented instruction, take the example of making tea. Represented as a list, the instruction for making tea might look like:



```
ACT RTE 1 LEGS        1/3
 .78
ALICE              .78/FL280
 .78           31.1 NM
BOYDD              .78/FL280
 .78           52.9 NM
CONEE              .78/FL280
 .78           37.7 NM
BAETT              .78/FL280
 .78           37.7 NM
AYMAN              .78/FL280
- - - - - - - - - - - - - - -
<RTE 2 LEGS        RTE DATA>
```

Flight 123 ... proceed direct to BAETT

Figure 1. The FMS experiment. The left of the display shows the keyboard and display contents of the actual FMS unit, together with a button to give up or indicate that the task is completed. The top right of the display shows the navigational display that can be used to verify the route. The bottom right of the display shows the current problem, and will display feedback after the participant has pressed “finish”.

1. Put water in kettle
2. Put kettle on stove
3. Put leaves in teapot
4. Wait until water boils
5. Pour water in tea pot

Now suppose you are a Martian and know nothing about making tea, but were taught the elementary steps in the above instructions. In that case you would be able to make tea in ideal circumstances, but you wouldn’t know what to do with a kettle already full of water. You also wouldn’t know that putting the leaves in the teapot is something that you can do, say, before you put the kettle on the stove. Although this may seem like a far-fetched example, more complicated recipes do have steps that seem arbitrary. Moreover, it mirrors instructions in many situations in which users have to operate some novel device or piece of software. Instructions in these cases are also often written as lists of steps to carry out without explanation of what these steps are for. Now consider the following version of the tea instructions, in which we specify pre- and postconditions of each step:

[empty kettle] put water in kettle [kettle with cold water]
[kettle with cold water] put kettle on stove [water boils]
[empty teapot] put leaves in teapot [leaves in teapot]
[water boils and leaves in teapot] pour water in teapot [have tea]

In this representation the instructions tell you when they should be applied and what they accomplish. They also allow more flexible behavior. For example, these instructions specify what to do when the kettle is already full of water, and they also allow putting the leaves in the teapot before anything else. Another useful aspect of these instructions is that if one of the steps is forgotten, it is much easier to reconstruct or guess the missing step on the basis of the pre- and postconditions. So if “put kettle on stove” is forgotten, the system can infer that it needs to find a step to get from “kettle with cold water” to “water boils” (and can even try to find alternatives if the stove is broken).

The fact that these extended instructions work so much better for a model led us to the hypothesis that extended instructions would also work much better for people, especially in domains where it is unclear what the function of individual steps is.

## The FMS Task

Many modern passenger airplanes use Flight Management Systems (FMS) to help control the airplane. On a routine flight, the FMS can perform almost the whole flight with the exception of take-off and landing. The task of the pilot is to supply the FMS with the right information and parameters to do its job, for example the load of the plane, but, most importantly, the route that it has to fly. This route consists of a list of waypoints that the plane has to follow from the source to the destination airport. Waypoints are sometimes specific radio beacons, but often are just points on the map with particular coordinates. Although the route in the FMS has both a vertical and a lateral component, the experiment focuses on the lateral part of the task, which is not unlike a route planned by programs like Mapquest.

Interacting with the FMS is typically learned as part of the pilot’s supplementary training when they first start flying a plane that has an FMS. Training consists of a phase in which procedures on the FMS are learned in the classroom, followed by a phase in which they are applied in a simulator. Procedures are specified as lists of steps to carry out, very much like the list representation of the tea example. Although 102 different procedures have been identified for the Boeing 777 FMS, the system we used for our experiment, knowledge of only around 25 procedures is needed for FAA certification, and therefore the training focuses on those procedures. The idea behind this is that the pilots can study and/or discover the remaining procedures on their own. Experience from training itself shows, however, that it is very hard for pilots to learn the required procedures, let alone discover any new procedures (Sherry, Polson, Fennell & Feary, 2002). Memorizing the procedures during the classroom phase of training turns out to be so hard that it is virtually useless for the second phase of training in the simulator. Pilot's troubles include problems with forgetting particular steps in a procedure, not knowing how to pick up a partially completed procedure, and poor generalization. For example, the procedure to fly towards a waypoint at a certain heading is identical to the procedure needed to land the plane (which is approaching the end of the runway at a certain heading), but pilots have great trouble executing the former while having no problems with the latter.

The problems outlined above are very similar to the problems that we had with models that used a linear representation of instructions. Improving the representation of instructions paid off for the model, so if our models are correct, changing instructions from a linear style to a more extended style should produce significant improvements in learning and eventual performance in humans as well.

## Experiment

### Task

Participants had to do a lateral navigation task, in which they had to change the route programmed in the FMS as directed by Air Traffic Control. They were first given some general background about the FMS and airplane routes, and were then taught two procedures: the *direct-to* procedure and the *resolve-discontinuity* procedure. The direct-to procedure specifies how to change the waypoint the plane is currently flying to, so the first waypoint on the list that specifies the whole route. Sometimes changing this waypoint required the specification of how the new waypoint connects to the rest of the route, for which the resolve-discontinuity procedure had to be used. Participants had to perform the task using a simulated FMS and navigational (NAV) display (Figure 1). For the instruction of the procedures there were two conditions, the list-style procedure and the operator-style procedure.

The list-style procedure was taken directly from the United Airlines training program:

**Direct-to:**

1. Press the LEGS key
2. Enter the desired waypoint in the scratchpad
3. Push the 1L key
4. If the word "discontinuity" appears on the screen, follow the procedure to remove discontinuities.
5. Verify the route on the Navigational Display
6. Press EXEC

The operator-style procedure not only told participants the steps that they had to take, but also their purpose:

**Getting to the LEGS page**
You can see what page you are on by looking at the top line of the window. If the word "LEGS" is on that line then you are on a LEGS page

If you want to change the route and you are not yet on the LEGS page, then press the LEGS key in order to go to the LEGS page.

**How to modify a waypoint**
The item in line 1 on the first LEGS page, displayed in magenta, is the waypoint you are currently flying to.

You can change this item, or any other waypoint, through the Line Keys next to them.

If you want to modify a waypoint, you enter the waypoint to replace it with into the scratchpad, and then press the line key corresponding to the waypoint you want to modify.

**How to confirm your results**
Use the NAV display to view the results of your modification. When you are satisfied with a modification, you can press the EXEC key to make it permanent.

In addition to the direct-to procedure participants were given instructions to resolve discontinuities.

### Participants

Thirty-one students from the Carnegie Mellon University volunteered to participate in the experiment (15 in the list-style condition and 16 in the operator-style condition). Volunteers were paid for their participation.

### Procedure

Participants first read through the general background information of the FMS task, and then studied the direct-to and resolve-discontinuity procedures for the condition that they were in. They then started with a series of warm-up trials that taught them how to operate the FMS interface, making sure that they had mastered the elementary steps in the procedures (e.g., entering waypoints into the scratchpad). The experiment proper consisted of three main blocks of trials, each consisting of 12 problems. The first three problems in each block were problems for which the direct-to procedure could literally be applied. The second set of three problems consisted of direct-to problems with a discontinuity, so both the direct-to and the resolve-discontinuity procedure have to be applied. Each of the final six problems in a block contained some complication, making it impossible to literally apply the procedures. These complications were one, or a combination of:

- One of the waypoints referred to in the problem would not be on the page visible in the FMS. Participants had to use the page-up/down keys to find them. Although the function of these keys was explained in the general background, they were not part of the procedures.
- The waypoint to be modified was not the waypoint that the airplane was currently flying towards, but one later in the flight plan. This was not covered by the procedures, and required some generalization.

## The Model

Before we will proceed to discuss the results of the experiment, we will first describe the ACT-R model. A key aspect of the model is that it assumes that not all instructions have been successfully memorized. For any step in a procedure there is a 25% probability that it has been forgotten. This is a somewhat crude approximation of forgetting that could have been handled by decay in base-level activation. We took an explicit percentage to have some more control over this aspect of the model. Steps are represented with a precondition, an action, and a postcondition. The list-style instructions are represented with dummy pre- and postconditions that are just used to link them together, while the operator-style instructions have pre- and postconditions that can be matched to the state of the interface. This allows the use of the same model for both types of instructions. For example, the first steps of the instruction would be

[start] Press LEGS key [state1]
[state1] Enter destination in scratchpad [state2]
[state2] Press 1L [state3]

in the list-style instruction, where start indicates the start of a problem, and state1 is a meaningless label, but

[not on legs page] Press LEGS key [on legs page]
[on legs page] Enter destination in scratchpad [destination in scratchpad]
[destination in scratchpad] Press 1L [route is modified]

in the operator-style instruction, where both the pre- and postcondition can be matched against the environment.

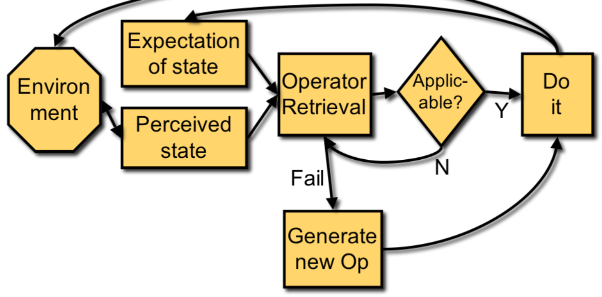


Figure 2. Outline of the model's operation.

The general operation of the model is summarized in Figure 2. On each step, the environment produces a perceived state of the world. In addition to the perceived state of the world there is an expectation of what the state should be. Initially this expectation is just "start", but afterwards it is set to the postcondition of the last operator. Both the expected and the perceived state now serve as sources of activation for the next step, operator retrieval, in which some operator is retrieved from declarative memory for the current task. The spreading activation from both states makes it likely that a relevant operator will be retrieved, especially if both states are in agreement with each other (which is only true for operator-style instructions). After an operator has been retrieved, productions check whether it is applicable in the current state. If it is applicable, the operator will be carried out, and the cycle restarts. If the operator is not applicable, a new operator is retrieved. If the operator retrieval process produces a retrieval failure (because it has ended up in an unfamiliar state, or when part of the instructions have been forgotten), a new operator will be generated. This operator will use the current perceived state as its precondition and carry out a randomly generated but currently applicable action. This operator will then be applied, after which the resulting state is added as a postcondition.

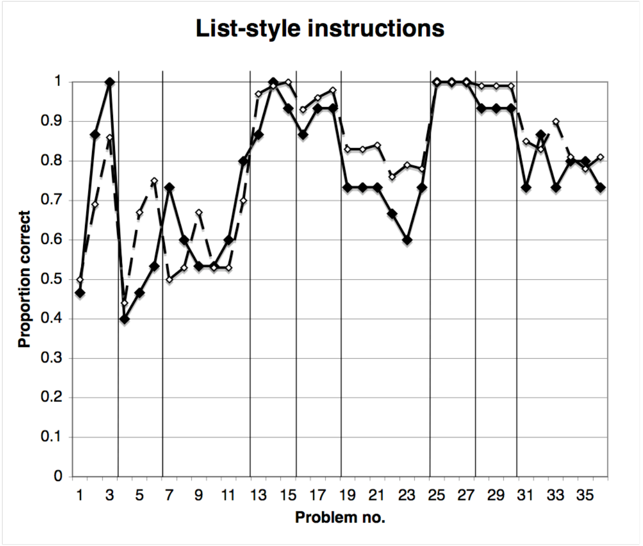


Figure 3. Proportion correct for the list condition. Data (solid line) and model (dashed line).

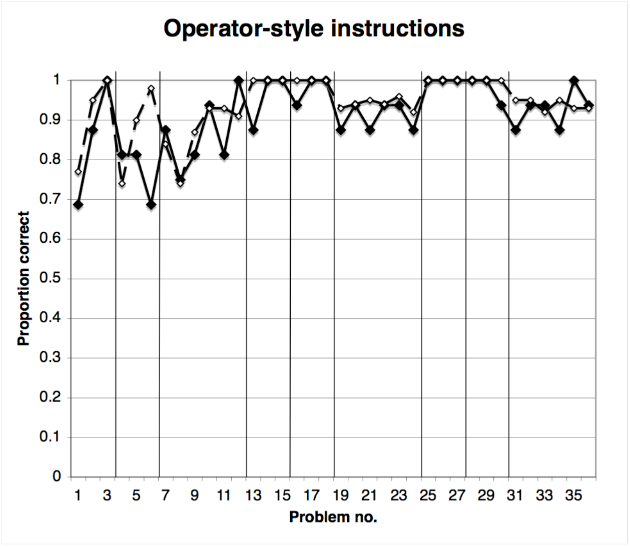


Figure 4. Proportion correct for the operator condition. Data (solid line) and model (dashed line).

Two processes will improve behavior with experience. First, there is the generation of new operators that fill in the gaps between forgotten steps of the instruction. Second, production compilation, ACT-R's rule learning mechanism, will learn new, task-specific production rules that will directly implement the relevant action for a particular state, bypassing the slow operator retrieval-and-test procedure. For example, a rule is learned that checks whether the current page is not the LEGS page, and immediately issues the motor commands needed to press the LEGS key, instead of going through the cycle of three production rules: (1) retrieve an appropriate operator, hopefully retrieving the operator to push the LEGS key, (2) check whether pushing the LEGS key is applicable, and (3) executing the manual command to press the LEGS key.

Two aspects of the model are not yet completely realistic and have to be fleshed out in future work: it can currently perfectly derive the perceived state from the environment, and can perfectly recognize how close a state is to achieving the goal.

## Results

Figure 3 and 4 show the accuracies of respectively the list- and the operator-style instructions, with the data in solid lines and the model results in dashed lines. The vertical lines in the graphs indicate boundaries between the difficulty levels of the problems: problems 1-3 are easy problems, problems 4-6 have a discontinuity, problems 7-12 are hard problems, 13-15 are easy again, etc.

Operator-style instructions lead to a significantly higher accuracy ($F(1,29)=7.086$, $p=.013$), especially for the harder problems, leading to a significant interaction between condition and problem difficulty ($F(1,29)=4.422$, $p=.044$). The model reproduces the main effects in the data. The list model is somewhat capable of overcoming the problems of the list-style instruction. The reason for this is that as soon as the model starts running into problems, exploratory strategies enable it to learn operator-style instructions. Eventually the list-style model ends with a mixture of learned list-style instructions and self-discovered operator-style instructions. The self-discovered instructions are often less general than the instructions given in the operator-style condition, making the model less flexible when it faces the hard problems.

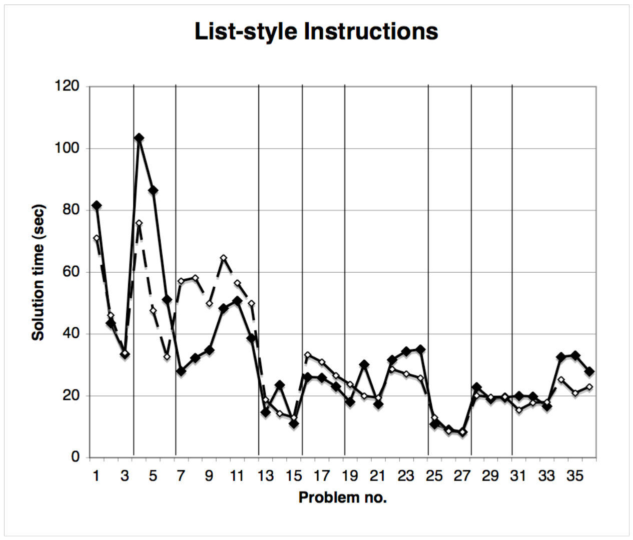


Figure 5. Solution time for the list condition. Data (solid line) and model (dashed line).

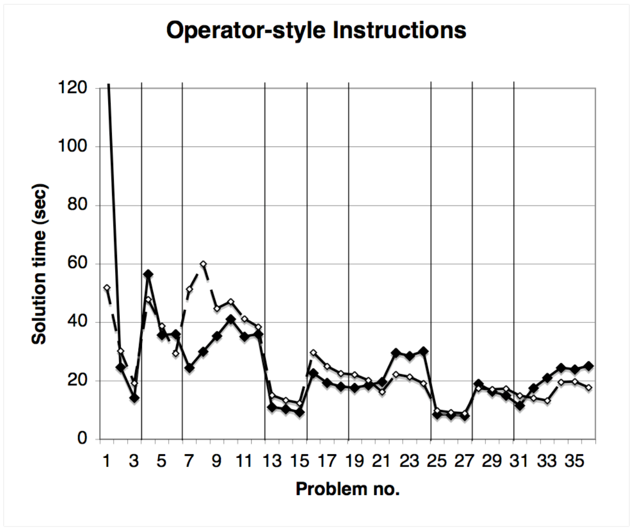


Figure 6. Solution time for the operator condition. Data (solid line) and model (dashed line).

Figure 5 and 6 show the average solution times for both conditions (for correct solutions). Mirroring the accuracy data, participants are significantly slower in the list-style condition ($F(1,29)=4.560$, $p=.041$). So, even when participants in the list condition find the solution, they need more time to do so.

During the initial stages of the experiment the model looses most of its time on retrieving operators that it cannot apply yet. This is particularly true for the list-style instructions, because the expected and perceived state are not in agreement (Figure 2). Time is also lost in trying to construct forgotten operators and getting out of error states produced by exploratory behavior. Eventually it will learn productions that directly implement the operators, leading to the fast performance that participants also exhibit.

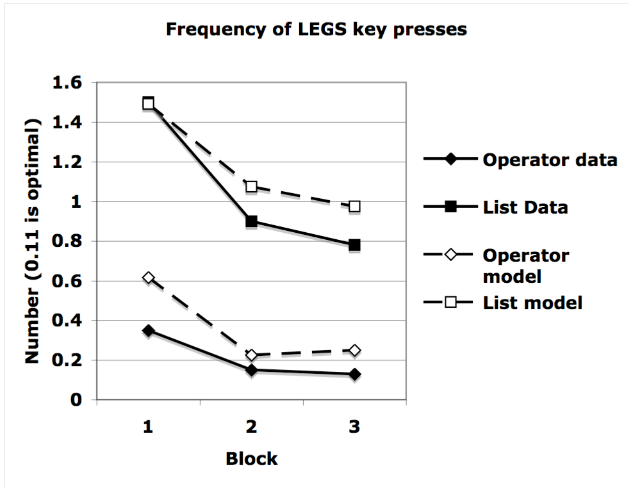


Figure 7. Average number of presses on the LEGS key per problem

We can also look at performance at a more detailed level. One of the problems of the list-style instructions is that it is often unclear what the purpose of a certain step is. For example, the first instruction is to push the LEGS key. If the FMS is already on the LEGS page, this step is unnecessary. In 32 of the 36 problems, the FMS already displayed the LEGS page at the start of the problem. This means that optimal performance entails pressing the LEGS key 0.11 times per problem on average. Figure 7 show the model and data for both conditions. Participants in the list-style condition press the LEGS key much more often than needed, while participants in the operator-style condition are close to the optimal level.

A second example is the use of the EXEC key. This key is normally used as the last step in a route-change procedure, because it commits the FMS to the change route. The List instructions (as taken from United Airlines) not only specify that the EXEC key has to be pressed at the end of the direct-to procedure, but also at the end of the procedure to resolve a discontinuity. As Figure 8 shows, this leads to extraneous presses of the EXEC key in problems with a discontinuity. Neither the participants nor the model follow the instructions to the letter, because the average number of key presses on the EXEC key is around 1.3 eventually, instead of the 2 that the instructions prescribe. As expected these extra presses on the EXEC key are virtually absent in the operator-style condition (Figure 9).

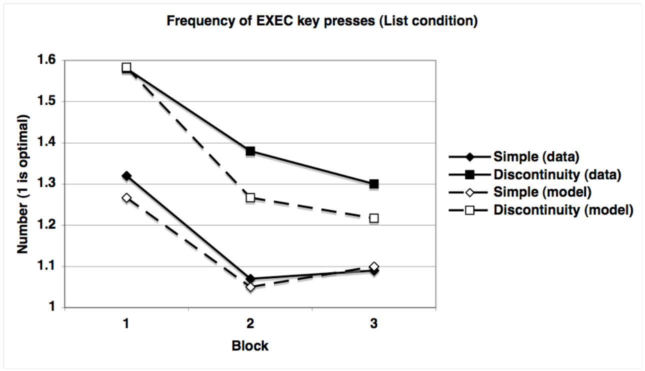


Figure 8. Average number of presses on the EXEC key in the list condition for problems with and without a discontinuity.

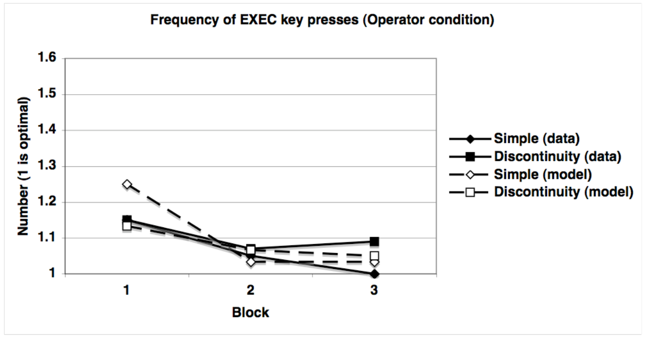


Figure 9. Average number of presses on the EXEC key in the operator condition for problems with and without a discontinuity.

## Conclusions

The model and experiment presented here show that instructions designed on the basis of a cognitive model can indeed produce significant improvements in performance, and that the improvements are what the model predicts. Although the model is not completely accurate for every single problem instance, it is sufficiently accurate to capture the differences between the two conditions. The newly designed instructions not only lead to faster learning, but also help the learner to go beyond the direct scope of the instructions and help generalization to different problems.

The components of this model find their roots in various other modeling projects in addition to the previous work we mentioned in the introduction, like impasse-based learning in Soar, classical AI planning and situated cognition. All these approaches combined, however, produce a model that learns to do a complex task from instructions that involves the simulation of a real device (and not a simplification).

## Acknowledgements

This research is supported by NASA grant NRA2-38169. We would like to thank Peter Polson and Karl Fennell for their advice.

## References

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

Anderson, J.R., Bothell, D., Byrne, M., Douglass, D., Lebiere, C. & Qin, Y. (2004). An integrated theory of mind. *Psychological Review, 111* (4), 1036-1060.

Card, K.C., Moran, T.P. and Newell, A. (1983). *The psychology of human-computer interaction*. Hillsdale, NJ: Erlbaum.

Kieras, D.E., Meyer, D.E., Ballas, J.A. & Lauber, E.J. (2000). Modern computational perspectives on executive mental processes and cognitive control: where to from here? In S. Monsell & J. Driver (Eds.), *Control of cognitive processes. Attention and Performance XVIII* (pp. 681-712). Cambridge, MA: MIT Press.

Lee, F.J. & Taatgen, N.A. (2002). Multi-tasking as Skill Acquisition. *Proceedings of the twenty-fourth annual conference of the cognitive science society* (pp. 572-577). Mahwah, NJ: Erlbaum.

Sherry, L., Polson, P., Fennell, K., Feary, M. (2002). Drinking from the Fire Hose: Why the Fmc/Mcdu Can Be Hard to Learn and Difficult to Use. Honeywell internal report C69-5370-022.

Taatgen, N.A. (2005). Modeling parallelization and flexibility improvements in skill acquisition: from dual tasks to complex dynamic skills. *Cognitive Science, 29,* 421-455.

Taatgen, N.A. (in press). The Minimal Control Principle. In W. Gray (Ed.), *Integrated Models of Cognitive Systems*. Oxford University Press.

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