Carnegie Mellon University
Research Showcase @ CMU

Department of Psychology | Dietrich College of Humanities and Social Sciences

8-2008

The Acquisition of Robust and Flexible Cognitive Skills

Niels A. Taatgen
Carnegie Mellon University

David Huss
Carnegie Mellon University

John R. Anderson
Carnegie Mellon University

Follow this and additional works at: http://repository.cmu.edu/psychology

Part of the Psychology Commons

Recommended Citation
.

This Article is brought to you for free and open access by the Dietrich College of Humanities and Social Sciences at Research Showcase @ CMU. It has been accepted for inclusion in Department of Psychology by an authorized administrator of Research Showcase @ CMU. For more information, please contact research-showcase@andrew.cmu.edu.

Running head: Robust and Flexible Skills

The Acquisition of Robust and Flexible Cognitive Skills

Niels A. Taatgen[12], David Huss[1], John R. Anderson[1]

[1]Carnegie Mellon University and [2]University of Groningen

Keywords: Control, Cognitive Modeling, Skill acquisition, Learning from instructions, Embedded Cognition

Abstract

We introduce a model of skill acquisition that strikes a balance between top-down and bottom-up control using a knowledge representation in which pre- and post-conditions are attached to actions. This model captures improved performance, defined as shorter solution times and lower error rates during the task. The bottom-up control element also allows the model to show increased flexibility to solve similar problems and robustness against unexpected events. In two experiments using a complex aviation task, we contrasted instructions that explicitly stated pre- and postconditions with conventional instructions that did not. The augmented instructions led to better and more robust performance than standard instructions, especially on problems that required transfer. The results of Experiment 1 were fit by our model, which was then successfully used to predict the results of Experiment 2.

The Acquisition of Robust and Flexible Cognitive Skills

Humans have the remarkable ability to acquire almost any skill given suitable instructions and practice. Models of skill acquisition (Newell & Rosenbloom, 1981; Anderson 1982, 1987; Logan, 1988) have mainly focused on two aspects of skill acquisition: improvement in speed and reduction of errors. Despite the focus on speed and reduction of error, *expertise*, which can be considered the end product of skill acquisition, is not only associated with speed and accuracy, but also with flexibility and robustness (e.g., Chi, 2006). Flexibility refers to the ability to apply a skill to new problems that are different from the problems that served as the basis for training. Robustness is associated with the ability to protect skilled performance from various disturbances, including unexpected events, interruptions, or changing demands. It also includes the ability to perform a skill in parallel with other tasks, without any obvious need for complex strategies to schedule resources between multiple tasks. The goal of this article is to discuss a model that can explain the improvement in speed and the reduction in errors, but also explain why flexibility and robustness improve with practice. The key to understanding this improvement is to assume that cognitive task control has both an internal or top-down component, and an external or bottom-up component. The bottom-up component is crucial to enable the model to adapt quickly to changes in the task environment without needing extra knowledge or inference capabilities.

Overview of Models of Skill Acquisition

Although many models of skill acquisition have been proposed, most of them share a mechanism in which the model starts with general knowledge, and through experience gains more specialized knowledge. This specialized knowledge has the advantage of faster memory-access, and, assuming the specialized knowledge is correct, a reduction in errors.

Logan's (1988) instance theory assumes people start out with a general-purpose algorithm to solve the problem. Each correct solution is stored as an instance in memory. For each new problem, there is a potential race between instances in memory and the general-purpose algorithm: whichever produces the answer first wins and determines behavior. Instance retrieval is generally faster than the algorithm, which explains the speedup in performance as instances accumulate.

Newell and Rosenbloom's (1981) chunking mechanism explains skill acquisition by specializing general problem-solving strategies. The model starts out with a general strategy to solve the problem, which may entail setting subgoals to solve partial problems. Whenever a partial problem is solved, a new rule or *chunk* is created, which can solve a partial problem for that specific case on future occasions. An even more fine-grained version of this idea is used in the ACT-R architecture (Anderson et al, 2004). ACT-R has a rule learning mechanism called *production compilation* (Taatgen & Anderson, 2002), which combines any two rules that have been used in sequence into one new rule. Any memory access of these rules is substituted into the new rule, essentially making it a specialization of the original rules.

Despite many differences in details, the three models have a similar account of the general characteristics of skill acquisition: they all assume an initial general strategy that can solve the problem and they all explain the speedup in performance and the reduction in errors by specialization and more efficient use of that strategy. The main difference is the grain-size of specialization: in instance theory it is the problem as a whole, in chunking it is at the level of individual sub-tasks (which can include several steps), and in production compilation it is the level of individual steps in problem solving.

None of the three models addresses robustness and flexibility directly: the nature of the learning process implies specialization, which generally means that the learned knowledge cannot be used for cases that have not been seen before. All three models do have some means to infuse the specialization process with some generalization, which can potentially help increasing flexibility. In instance theory, a similar but not identical instance can be retrieved, and applied to the current situation (Lebiere, Wallach & Taatgen, 1998), in chunking a constant can be replaced by a variable (Newell, 1990), and in production compilation the general strategy of analogy can be specialized to achieve generalization (Taatgen & Anderson, 2002). Although generalization can improve flexibility because it helps covering more different situations, it can also lead to overgeneralization, which leads to the choice of inappropriate actions that reduce robustness instead of improving it. We will take a different approach to achieving flexibility and robustness by focusing on the control aspects of task performance.

The role of control in complex task performance

Models of complex task performance traditionally structure the knowledge for the task in a sequence of steps, each of which transforms the current state of the system into a new state. These steps can be structured in lists, or, in the case of complex tasks, a hierarchy of lists. The GOMS system (Card, Moran & Newell, 1983) is the best-known example of this method. Although hierarchical task representations can offer insight in the structure of a task, they are not necessarily appropriate as a starting point for understanding human task representations. A cognitive model that is driven by a straightforward hierarchical control structure is often very inflexible: it cannot handle interruption and recovery, unexpected external input, slight deviations in the task, and it has no account for the fact that well-practiced tasks can be carried out in parallel with other tasks (Taatgen, 2005). There are two possible solutions to this problem.

A first solution is to add knowledge and strategies, either general or specific for the task. For example, Kieras, Meyer, Ballas and Lauber (2000) propose a *general executive* to handle multi-tasking. The general executive grants each of the sub-tasks access to various cognitive resources. However, they conclude that a general executive does not lead to the efficient behavior humans show after sufficient training, creating a need for *specialized executives* for particular combinations of tasks. More in general, the addition of general knowledge cannot handle all special cases, which is known as the *frame problem* in artificial intelligence (McCarthy & Hayes, 1969). However, there are often too many special cases to be able to add all of them to a model.

Although adding knowledge is probably an explanation of flexibility to some extent, it is not suitable as a general solution to all flexibility issues, because it would require specific knowledge for all possible unexpected events. A second solution is the opposite of the first, because it aims to reduce the amount of specific knowledge for a task. The solution starts with the observation that control does not always have to be internal or top-down, but can also be derived from the environment. When executing a task, it is often not necessary to plan and control every step, because the environment cues the appropriate next step (we call this bottom-up control). As a consequence, internal or top-down control can be reserved for cases in which the outside world is ambiguous. Simplifying internal control increases flexibility and robustness, because with fewer internal states less knowledge is needed to handle exceptions and unexpected events. Taatgen (in press) has called this the *minimal control principle*:

> People organize their behavior for a task such that the cognitive model that corresponds to this behavior has the minimal number of control states.

To illustrate the principle we will look at Larkin's example of making coffee (Larkin, 1989). When analyzed in detail, making coffee turns out to be a complicated task that consists of

many mutually dependent steps. Larkin observes that people are able to navigate through these steps almost without effort, and are also able to adapt their plan on the fly. The key observation she makes is that individual steps are not triggered by a planning process, but rather by conditions that can be perceived in the world, e.g., an empty filter holder cues placing a filter in it. Informal investigation of errors people make in preparing coffee revealed that errors are often related to aspects of the task that cannot be observed directly, e.g., forgetting to fill an opaque water reservoir. This study suggests that people do not mentally organize the steps required to make coffee as a list. Instead the conditions under which a step can be carried out plays an important role in organizing knowledge and selecting the next steps. For example, the step to put ground coffee in the filter is not triggered by the previous step of grinding coffee, but by the fact that there is an empty filter in the holder, and ground coffee in the grinder. Many of these conditions can be perceived, while others have to be remembered (e.g., whether the opaque container has been filled).

Representing task knowledge

Larkin's example shows that it is only necessary to maintain an internal representation of the problem state as far as the external world is ambiguous. In order to exploit this reliance on bottom-up control, a representation of task knowledge that links actions in a hierarchy is not the right solution, because this hierarchy implicitly creates internal states for each action. Instead, a representation is needed in which the order is left largely unspecified, but in which each step has triggering conditions (preconditions) that can be perceived in the world.

Recent research by Catrambone and Yuasa (2006) also supports this viewpoint. In their experiment participants had to formulate database queries, and received instructions that either emphasized the connection between conditions and action, or mainly elaborated on just the

conditions. They found that the instructions connecting condition and action produced significantly lower error rates.

In this paper, we take this idea a step further and add to each step a representation of its expected outcome. Encoding task knowledge in terms of condition-action-outcome triples results in what we call an *operator* representation because of their similarity to representations used in Artificial Intelligence (STRIPS operators, Fikes & Nilsson, 1971) and in the early work of Newell and Simon (Newell & Simon, 1963). Although STRIPS has also been used to control a robot, most of the earlier AI systems matched preconditions only to an internal representation of the problem, giving rise to the criticism that they only modeled cognition "in the head". In our approach the preconditions of an operator can be matched to both internal states and to what is perceived in the world. However, matching the precondition to conditions in the world produces behavior with maximal flexibility.

The basis for this representation is our previous work on dual tasking (Taatgen, 2005). When two tasks have to be performed at the same time, and each of the two tasks has its own list of steps, it is a challenge to interleave these steps properly. This is an example of robustness in which two separately practiced tasked are combined. Our attempts to build an internal cognitive model of perfect time sharing (using the results from Hazeltine, Teague, & Ivry, 2002, and Schumacher et al., 2001) failed, not only because of the complexity of interleaving the two tasks, but also because the model could not adjust to fluctuations in the timing of perceptual and motor processing. A model in which the steps were triggered by external stimuli and the completion of internal memory retrieval processes did allow the model to achieve perfect time sharing in the same way as human participants (Anderson, Taatgen, & Byrne, 2005; Taatgen, 2005). These perfect time-sharing models showed that a bottom-up control structure was able to explain

perfect time-sharing without the need of a general or specialized executive. Although the perfect time-sharing models are good examples of how robustness with respect to multi tasking can be achieved with simple models, the tasks were too simple to address other aspects of flexibility and robustness.

In this article we will present a cognitive model of a complex task to show how a minimal top-down control can explain other aspects of flexibility and robustness, and explore the additional mechanisms needed to achieve them. The model aims to explain how missing knowledge is filled in, how problems that do not match the instructions exactly can nevertheless be solved, and how partially completed problems and error states can be handled. The model uses an operator representation, which allows the model to select actions directed by both top-down and bottom-up control.

Before analyzing our model in detail, we describe an experiment in which we tested our assumptions about the relationship between task structure and the robustness and flexibility of the subsequently acquired skills by manipulating the instructions given to participants. The experiment has two conditions: one ("list condition") in which participants received instructions as lists of steps to carry out, and a second one ("context condition") in which instructions were presented with actions augmented with the conditions in which they have to be carried out. Because the context condition better fits the operator representation that we assume people use we expect that it will lead to a flexible control strategy in which the environment prompts the next step. The list representation on the other hand, promotes a control strategy that is mainly top-down, because one has to keep track of which steps have been carried out.

We expected the context condition to lead to fewer errors and faster solution times. In addition, we expected that this difference would be larger for problems that require some

adaptation or generalization of the original instruction. We have constructed a cognitive model of performing this task using the ACT-R architecture (Anderson et al., 2004). A single model interprets both types of instructions, making detailed predictions about solution time and accuracy under each condition.

We replicated and extended our findings in a second experiment, which also included partially completed problems, sometimes containing an error, as a further test of the model's robustness. Because the instructions for experiment 2 were identical to experiment 1, this allowed a prediction of the outcome of experiment 2 on the basis of the model of experiment 1.

The Task Domain: Flight Management Systems

The task used in both experiments is derived from automation in modern airplanes. Many passenger airplanes use Flight Management Systems (FMS) to help pilots 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, e.g., the load of the plane, but, most importantly, the route 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 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 the kind of routes that are produced by route planning services for cars.

Interacting with the FMS is typically taught as part of the pilots' supplementary training when they 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. 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. Pilots' 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 involves 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. Fennell, Sherry, Roberts, & Feary (2006) found that steps that cannot be inferred from the interface are the main source of errors, which is another indication that people's internal representation of instructions tends to be triggered by external events.

The FMS task is a very suitable task for exploring flexibility and robustness, because the starting point is a training system that for many pilots leads to inflexible and brittle skills. From the perspective of our minimal control principle this is perfectly understandable: if instructions are learned as lists of steps, the individual steps cannot be related to the current environment, and therefore cognitive control has to be completely internalized.

General task description for both experiments

For the purposes of the two experiments we chose two FMS procedures participants had to learn and carry out. Both procedures are part of *lateral navigation*. Lateral navigation involves

planning and modifying routes. A route is a sequence of waypoints (points on the map) that the plane will follow, and is typically programmed into the FMS before the flight starts. However, the route is often changed during the flight. A possible reason for such a change is that when traffic is light Air Traffic Control allows the plane to skip a few waypoints, and fly directly to a waypoint further in the flight plan. The procedure to make this modification is called *direct-to*. Another reason for changes in the flight plan is bad weather. This sometimes requires the pilot to change his heading to a new waypoint that was not previously on the flight plan, and proceed with the flight plan after that new waypoint. This change requires two procedures: the *direct-to* procedure to change the waypoint that the plane is currently heading for, and the *remove-discontinuity* procedure, which tells the FMS how to connect the newly entered waypoint to the rest of the flight plan.

Participants had to carry out route modifications on a simulated FMS, consisting of a keyboard to enter information, a display listing part of the current route, a navigational display showing a map with waypoints and the current route, and a pane that shows the current task (Fig. 1). Participants had to operate the keyboard by clicking on the keys with the mouse. The simulation was faithful to a real FMS in all aspects relevant for doing the tasks.

There were two conditions for the instruction of the procedures: "list procedures" and "context procedures." List procedures, consisting of numbered lists of steps, were adapted from the United Airlines training program. The context procedures not only told participants the steps that they had to take, but also the conditions for carrying out a step, and what its result was. Table 1 lists both procedures in both styles. Fig. 2 illustrates how the *direct-to* procedure is carried out. The experiments consist of series of problems that participants have to solve, i.e., directives of Air Traffic Control that have to be entered in the FMS. In some of the problems the

learned procedures could be carried out literally, but in some more complicated problems participants had to make modifications to make it work. Our basic hypothesis is that the extra information provided by the context procedures will enable participants to solve those more complicated problems that go beyond the explicitly instructed *direct-to* and *remove-discontinuity* procedures.

Experiment 1

Method

Participants.

Thirty-one students from Carnegie Mellon University were paid for participation in the experiment (15 in the list condition and 16 in the context condition).

Procedure.

Participants first read through the general background information of the FMS task. They then started with a series of warm-up trials that taught them how to operate the FMS interface. These warm-up trials, taking about 10 minutes, consisted of typing in text on the keyboard, copying text from a line on the FMS to the scratchpad, and copying text from the scratchpad to a line. All of the text on the display during the warm-up was unrelated to the real FMS task. Participants then studied the *direct-to* and *remove-discontinuity* procedures for the condition that they were in for 5 minutes.

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 (easy problems). The second set of three problems consisted of problems in which a new waypoint had to be entered, making it necessary to apply both the *direct-to* and the *resolve-discontinuity* procedure (medium problems). Each of the final six

problems in a block contained some complication, making it impossible to apply the procedures literally (hard problems). These complications were one, or a combination of the following:

- 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.

Half of the hard problems only required using the *direct-to* procedure, the other half required both procedures. The whole experiment lasted approximately one hour. Once the main phase of the experiment had started, participants were not allowed to refer back to the instructions. If participants were unable to solve the first problem the experimenter would reiterate the relevant part of the procedure to help them. No help was offered at any later time in the experiment.

Results

Fig. 3 shows the proportion correct for each problem in the two conditions. The average correctness is 77.0% in the list condition, and 91.1% in the context condition. An analysis of variance with block and problem difficulty as within subject factors, condition as between subject factor and subject as random factor shows a main effect of condition ($F(1,29)=7.09$, MSE=1.85, $p=0.013$) and a main effect of block number ($F(1,29)=56.1$, MSE=2.01, $p<0.001$), and a main effect of problem difficulty ($F(2,58)=7.12$, MSE=0.42, $p=0.002$). Furthermore, there is an interaction between condition and block ($F(1,29)=6.7$, MSE=0.24, $p=0.015$), an interaction between block and problem difficulty ($F(2,58)=6.6$, MSE=0.28, $p=0.003$), an interaction

between condition and problem difficulty ($F(2,58)=3.43$, MSE=0.20, $p=0.039$), and no three-way interaction. This indicates that correctness is significantly lower in the list condition, and that there is substantial learning between blocks. The interaction between condition and problem difficulty indicates the instruction type has different effects on accuracy for different levels of difficulty. Welch t-tests comparing the two conditions for each type of problem reveal that the difference can mainly be explained by the hard problems: for the easy problems there is no significant difference in difficulty (list condition: 90.4% accuracy, context condition: 93.8% accuracy, one-sided $t(28.8)=1.15$, $p=0.13$), for the medium problems the difference is significant (list condition: 77.0% accuracy, context condition: 91.0% accuracy, one-sided $t(18.9)=1.99$, $p=0.031$), while for the hard problems the difference is highly significant (list condition: 70.4% accuracy, context condition: 89.9% accuracy, one-sided $t(17.9)=2.6$, $p=0.009$). These results confirm our prediction that the benefit of context instructions would be restricted to those problems that required participants to go beyond the actual procedures instructed.

The average solution time for correctly solved problems in the list condition is 28.7 seconds, in the context condition 24.8 seconds. An ANOVA of the log solution times of correctly solved problems with block as within subject factor, and condition and problem difficulty as between subject factors and subjects as random factor shows a main effect of condition ($F(1,29)=5.85$, MSE=3.91, $p=0.022$), a main effect of block ($F(1,29)=231.9$, MSE=35.8, $p<0.001$), and a main effect of problem difficulty ($F(2,58)=130.9$, MSE= 10.5, $p<0.001$). In addition, there is an interaction between block and problem difficulty ($F(2,58)=31.7$, MSE=4.48, $p<0.001$), but no other interactions. Fig. 4 shows the average solution times for the correct trials in both conditions. Unlike accuracy, there is no interaction between instruction and problem difficulty for latency and, as we will see, this is a prediction of our model.

A Model of Learning from Instructions

The experiment shows that the manipulation derived from a context representation of instructions leads to better performance, especially for hard problems. We have built a cognitive model to fit the outcomes of the experiment in order to improve our understanding of the reasoning process and of how flexibility and robustness can be achieved in complex tasks. It also serves as a basis to make predictions for future experiments, more specifically experiment 2[1]. The general approach is to have a single model that can take two representations of instructions, list and context. The model has been built in ACT-R (Anderson et al., 2004), and is a further development of our work on learning from instructions (Anderson, Taatgen, & Byrne, 2005; Taatgen, 2005; Taatgen & Lee, 2003).

The ACT-R architecture

One of the basic assumptions of the ACT-R architecture (Anderson et al., 2004) is that there are two long-term memories: *declarative memory* and *procedural memory*. Declarative memory is used to store facts and experiences, and is basically passive: it retrieves memories upon request. Procedural memory contains condition-action patterns, productions, which map sets of internal and external states onto actions. These productions play an active role, because as soon as the production's conditions are fulfilled it triggers an action (assuming it wins the competition with other productions whose conditions are fulfilled). Fig. 5 shows an overview of the architecture, with procedural memory as the central component, surrounded by modules that are either part of internal cognitive processing or communicate with the outside world. Although procedural memory is the central component, its scope is limited: it can only access the end product of processing in the modules as made available in *buffers*. For example, productions

cannot match any arbitrary fact in declarative memory, but instead only the fact that is present in declarative memory's buffer. Buffers therefore serve the role of communication ports between the different components of the cognitive system.

We will not discuss all the details of the ACT-R architecture here, but focus on the mechanisms in the architecture that are important for the model, and discuss them along with the main features of the model.

Representation of Instructions

Although many models based on productions (in ACT-R, but also in other architectures) start with a task representation in the form of a set of productions, this is not a plausible account of how new skills are acquired. As was already pointed out in early work with the ACT* architecture (Anderson, 1982), it is much more likely that instructions for a new task are first stored in declarative memory. These instructions are retrieved step by step by productions, and are then interpreted and carried out by other productions.

A main problem in the interpretation process is to decide which step to do next. As pointed out in the introduction, the easiest way to decide this is to make the individual steps of an instruction into a list, and retrieve and perform these steps in order. The alternative is to use preconditions to determine which step to do next, which was the basis for context instructions. This representation is augmented with a postcondition that specifies the expected outcome of a step.

In order to use the same model for both conditions of the experiment, we used a representation of instructions that contains a precondition, an action, and a postcondition. In case of the list instructions, the pre- and postconditions are symbols whose only purpose is to link the instructions together in a list. In the context condition the pre- and postconditions are

representations that can be matched to the state of the world. Table 2 lists the representations for the two conditions.

The Basic Decision Cycle

Fig. 6 illustrates the basic decision cycle of the model. In the first step the model perceives the current state of the environment, in this case the state of the FMS, producing the *perceived state*. Besides a perceived state, there is also an *expected state*, which is set to the postcondition of the previous operator (the value is initially set to START). Based on both the perceived and expected state, an operator is retrieved from declarative memory. This retrieval is based on ACT-R's memory activation: the operator with the highest activation will be retrieved (see Anderson et al., 2004 for the details). For the purpose of this model, the important factor in this process is that operators that share many attributes with the current expected and perceived state receive extra activation, and are therefore considered first. Because the retrieval process is noisy, the retrieved operator is checked for applicability. If it cannot currently be applied, a new operator is retrieved until an applicable operator is found, or until the retrieval process fails.

When the operator is applicable, it is executed resulting in an action that leads to a change in the environment, in the perceived state, and in the expected state. The cycle is repeated until the goal is reached or time runs out (the model gives up after 120 simulated seconds).

Forgetting and Discovery of Operators

One of the problems in learning from instructions is that instructions are forgotten easily. People have some ability in handling situations where they have forgotten some of the instructions. Moreover, for the harder problems in the FMS task participants need to discover some operators themselves. To simulate the aspect of forgetting operators, each operator in the model had a 25% probability of being forgotten, i.e., not being present in declarative memory at

the beginning of the simulation. Forgetting operators leads to situations in which the model does not know what to do, i.e., the retrieval attempt of an operator fails. The model uses a relatively simple strategy to handle the situation: it takes a random action, and tries to perceive what the result of that action is. It then judges whether this action has brought it closer to the goal. If that is the case, it will create a new operator with the old state as precondition, the random action as action, and the new state as postcondition. Random actions were drawn from the following set: pressing the LEGS or RTE key, typing any of the waypoints mentioned in the problem, pressing any line key with any of the waypoints mentioned in the instructions, or the line key below it, pressing a line key with the term "discontinuity" or the line key below it, pressing the "Erase" key, pressing the "CLR" key, or checking the navigational display. The model's judgment of whether an action would bring it closer to the goal was based on the number of steps from the goal. Although it is likely participants have a sense of whether an action bring them closer to the goal, the model's perfect knowledge is an oversimplification.

Some of the participants did in fact use a guessing strategy to try out new actions. It is, however, likely that there are other strategies to find a next action, for example based on means-ends analysis. The simple guessing strategy turned out to be enough for the model to learn the missing steps.

Learning

Although ACT-R has a number of learning mechanisms, the mechanism that explains most of the learning in the model (in addition to the discovery of new operators) is *production compilation* (Taatgen & Anderson, 2002). Production compilation encodes operators directly into productions, making it unnecessary to retrieve and test operators from declarative memory. The basic process is very simple: if two productions fire in sequence, they are combined into a

single new production that is added to procedural memory. If the first of these rules makes a request to declarative memory, and the second rule uses the retrieved fact to perform an action, the retrieved fact is substituted into the new rule. Table 3 shows an example of this process, with two rules from the model translated into pseudo-English. In the example, the two original productions implement part of the operator interpretation system: the first production initiates the retrieval of an operator for the current task, and the second production presses a key after checking whether the preconditions have been fulfilled. The newly learned production, which combines the two original rules and the retrieved operator, immediately recognizes the current state, presses the key, and sets the appropriate expected state. The advantage of the new rule is that it can bypass the retrieve-operator and check-applicability cycle, and therefore produces considerable speed-up and reduction in errors.

The first time a rule is created, it receives a negative utility value. ACT-R uses the utility values of rules to decide which rule to use if there are multiple candidates. In the case of a new rule it has to compete with the rule that it originated from (the “old” rule), so a new rule initially has no chance to win the competition with the old rule. However, each time the rule is recreated, its utility value will be increased to approximate the value of the old rule. If the utility of the old and the new rule are close enough, the new rule has some probability of being used, and start gaining its own experiences. The consequence of this is that new rules are only introduced slowly, which is consistent with the notion that skill acquisition is slow. The learning speed is controlled by a learning parameter that determines how fast the utility of the new rule converges with the utility of the old rule.

Model fits

The model was run 150 times for each of the two instruction-sets, after which the accuracies and solution times for correct trials were averaged. Three parameters were estimated to fit the data: the retrieval threshold, which determines the minimum activation needed to retrieve facts from declarative memory, the learning rate, which affects how quickly new productions can compete with old rules, and visual attention latency, which determines how long it takes to perceive the current state of the environment. The model fits are shown alongside the data in Fig. 3 and 4. The model fits the data quite well (R^2=0.90 for the accuracies and R^2=0.89 for the log latencies) with some exceptions, which we will discuss shortly.

The model explains superior performance in the context condition as due to basically two factors. First, attempts to retrieve operators are more successful in this condition. In the list condition the perceived state (e.g., being on the LEGS page) is different than the expected state (e.g., being in state *Direct 1* after pressing the LEGS key, see Table 2). Only the expected state can be used to determine the next step, because the instructions make no reference to a real-world state. In the context condition the expected and perceived states are generally the same and therefore both help activate the appropriate operator through spreading activation. Therefore operator retrieval in the list condition will fail more often than in the context condition.

A second factor is that the model is better able to cope with gaps in its knowledge in the context condition. These gaps can occur because of failures in retrieval or, in the difficult problems, because the model finds itself in situations for which it does not have operators. When it finds itself in such a situation the model must guess an action. The model with a context representation is much more likely to be able to pick up successful operation application after such a guess because it can recognize operators relevant to the state that appears after the guess.

For instance, suppose the model has forgotten the step to press 1L (the fourth line in Table 2). It can either guess the correct action or not:

(a) Suppose it guesses its action correctly, and presses 1L. It now perceives that it is in the state *modification done*. On the next cycle it can retrieve the operator that has *modification done* as its precondition and continue on with the rest of the procedure. The list condition model knows it is in state *Direct 2*, and when it guesses the 1L action correctly, will discover that this brings it in state *modification done*. It now has no knowledge on what to do in that state, because it only has an operator that tells it what to do in state *Direct 3*. It therefore has to again guess what to do next.

(b) Suppose the model makes a wrong guess. In the context condition it can sometimes still recognize the state that it ended up in, and retrieve operators applicable to that state. In the previous example, if the model's guess would be to press the CLR key, the scratchpad would be cleared. The model with context instructions would immediately know that it has to enter the waypoint again, while the model with list instructions would have no information on how to proceed.

This ability to continue on after a knowledge gap is much more important in the difficult problems because there are more knowledge gaps. This is what accounts for the interaction between instruction and difficulty.

Interestingly, at knowledge gaps in the list condition, the model discovers operators that are context-style and not list-style. Eventually the list-condition model acquires a mixture of list-style and context-style operators. Thus, with practice the knowledge representation in the list condition becomes more and more like the knowledge representation in the context condition.

It is also worth noting that context instructions produce better performance because they skip unnecessary steps. For example, the first step 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. Fig. 7 show shows the number of presses of the LEGS key in the model and the data for both conditions. Participants in the list condition press the LEGS key much more often than needed, while participants in the context condition are close to the optimal level. The model correctly fits these proportions.

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 route change. 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 Fig. 8 shows, this leads to extraneous presses of the EXEC key in problems with a discontinuity. In the list condition, neither the participants nor the model follow the instructions to the letter, because the average number of key presses on the EXEC key condition for discontinuities is around 1.3 eventually, instead of the 2 that the instructions prescribe. The model learns to skip the extra EXEC key when it forms its own operators during exploration at knowledge gaps. As expected these extra presses on the EXEC key are virtually absent in the context condition. The model also correctly fits these data.

Experiment 2

An advantage of a cognitive model is that it can be used to make predictions. In order to test the predictive power and robustness of the model, we designed a second experiment that has

identical instructions as the first experiment, but with a new level of complexity in the actual tasks. The model predictions were generated before the actual data were collected.

One aspect of real-life interaction between pilots and the FMS that is not apparent from the way instructions are formulated is that pilots often share doing tasks on the FMS. It is possible that one of the pilots initiates a procedure, and than asks the co-pilot to take over. The co-pilot then has to assess the state of the system and has to decide which steps are still needed to reach the goal. This situation is similar to the situation in which a procedure is interrupted and resumed later on, because at the moment of resumption the state of the system has to be reassessed (assuming the pilot has not remembered this state). To simulate this in the experiment we included blocks of trials in which the task was already partially completed. To make it even harder for the participants, some of the half-completed trials contained an error. For example, in some of the problems the destination would already be in the scratchpad, sometimes a wrong destination.

Although the model was not designed to be able to handle these cases, it should in principle be able to solve them, especially the context-condition model, because its precondition matching strategy should be able to pick up on the current state of the system. Also, the model should be able to recover from problems containing an error at least some of the time, because it has to recover from the errors it makes during exploring unknown actions. On this basis we were able to make a prediction of the outcome of the experiment before actually conducting it.

Method

Participants.

Forty students from Carnegie Mellon University were paid for their participation in the experiment (20 in the list condition and 20 in the context condition).

Procedure.

The procedure was identical to the procedure of experiment 1, except that each of the three blocks now consisted of 24 problems. The first 12 problems were the same as experiment 1: three easy problems, followed by three intermediate problems and then six hard problems. Problem 13-24 in each block were partially completed. Half of the partially completed problems contained an error. The partially completed problems, either with or without error, consisted of equal proportions of easy, intermediate and hard problems. Participants were not informed that procedures could be partially completed, nor of the possibility that an error was already present, in fact, the instructions they received were identical to those of experiment 1. The experiment lasted on average 1.5 hours.

Results

Empirical results.

Fig. 9 shows the proportion correct for each problem in the two conditions. The average correctness in the list condition is 78.8% and in the context condition is 89.0%. An analysis of variance with block and problem difficulty as within subject factor, condition as between subject factor and subject as random factor show main effects of condition ($F(1,38)=4.5$, MSE=2.54, $p=0.040$) of block ($F(1,38)=35.3$, MSE=2.67, $p<0.001$), and of problem difficulty ($F(152,4)=26.0$, MSE=1.32, $p<0.001$). In addition, there are significant interactions between condition and problem difficulty ($F(152,4)=2.82$, MSE=0.14, $p=0.027$) and between block and problem difficulty ($F(152,4)=5.45$, MSE=0.24, $p<0.001$). Welch t-tests comparing the two conditions for each problem type are listed in Table 4. Overall the experiment replicates the results of experiment 1, and extends it by showing an effect of instruction on partially completed problems, although this effect is only significant in the case of the partially completed problems

with an error. The overall effect of instructional manipulation is not as strong as in experiment 1. This might be due to the fact that the experiment is twice as long, and the difference between the conditions tends to get smaller with practice.

An ANOVA of the log solution times was conducted with block and problem difficulty as within subject factor, condition as between subject factor and subject as random factor. The average solution time for correctly solved problems in the list-condition is 27.2 seconds, in the context condition 23.0 seconds but this difference is only marginally significant ($F(1,38)=3.60$, MSE=4.53, $p=0.065$). There is a main effect of block ($F(1,38)=120.9$, MSE=52.1, $p<0.001$), indicating a learning effect, a main effect of problem difficulty ($F(4,152)=115.2$, MSE=12.9, $p<0.001$), and an interaction between problem difficulty and block ($F(4,152)=27.4$, MSE=4.12, $p<0.001$). Fig. 10 shows the average solution times for the correct trials for each trial for both conditions combined.

Model prediction.

The prediction of the model is shown alongside the data in Fig. 9 and 10 and in Table 4 ($R^2=0.74$ for the accuracies and $R^2=0.76$ for the log latencies). A first noteworthy result is that the unmodified model is able to do the new problems at all, because it was not designed to solve them. This shows that the model indeed exhibits the flexibility and robustness that we were seeking. For the partially completed problems without error the predictions are very accurate. For the partially completed problems with an error the model predicts the list condition accurately, but slightly underestimates performance in the context condition. This is probably due to the fact that the model has no real error recovery strategies: it will just start trying things out when it is in an unrecognized state. Participants in the context condition might be better able to recognize an error state and therefore also better able to cope with one. The latency

predictions in Fig. 10 are also quite accurate, especially the partially completed problems of which we had no previous data. The only exception is an underestimation of the solution times for the first few problems (similar to the underestimation in experiment 1).

Conclusion

In this article we presented a model of skill acquisition that takes into account that cognitive control is both internal, in terms of an internal control state that keeps track of progress, and external, in the sense that the environment can prompt the next action. The model improves with practice by acquiring new knowledge, and by transforming task knowledge that is initially declarative into procedural knowledge. The flexibility of the model is derived mainly by offloading control to the environment, which leads to knowledge that is more general than knowledge relying on internal control states. The model offers a solution to what many have indicated as an ignored aspect of problem solving: information is in the world as well as in the head (e.g., Hutchins, 1995). A context-style task representation allows people to offload control to the world: given a proper understanding of the task they can select their actions not only on the basis of an internal state, but on the state of the world. This view of interaction with the world, although it is has its basis in symbolic paradigms for understanding cognition, is consistent with the embodied cognition paradigm (Brooks, 1991; Clark, 1997).

The model assumes that the representation of individual steps that make up a skill consists of an action with pre- and postconditions. This implies that instructions that are formulated in these terms should lead to better performance than instructions that just list the actions. Our two experiments showed that context instructions lead to better performance than list instructions in terms of accuracy, solution time and flexibility. The added flexibility is evidenced in problems that required generalization (the hard problems) and in problems in which

the task had to be picked up in the middle, possibly with an error (the copilot problems). Nevertheless, even when given list instructions both the model and the participants developed flexibility, only slower and less complete. The model explains this by an exploration process in which operators are discovered, because their pre- and postconditions are specified in full and can be related to the environment. One recommendation from this research is that the standard list procedures used in pilot training should be replaced by procedures in which the conditions and results of each action are explained explicitly.

Besides empirical evidence we have presented a model fitted to the data of Experiment 1. This model, in which the only difference between the two conditions was the representation of the instructions, could reproduce the accuracy and solution time data very well, and was also able to explain some of the details on how often particular (the LEGS and the EXEC) keys were pressed. The model generalized to novel situations: it predicted the outcome of Experiment 2, which contained partially completed trials the model was not designed for. The model not only solved these trials, it did so with accuracy and solution time comparable to those of participants. We believe that being able to make predictions is an important property of cognitive models, because it counters some of the criticisms that cognitive models can fit any dataset given enough free parameters (Roberts & Pashler, 2000).

References

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

Anderson, J. R. (1987). Skill acquisition: Compilation of weak-method problem situations. *Psychological Review, 94*(2), 192-210.

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

Anderson, J. R., Taatgen, N. A., & Byrne, M. D. (2005). Learning to Achieve Perfect Time Sharing: Architectural Implications of Hazeltine, Teague, & Ivry (2002). *Journal of Experimental Psychology: Human Perception and Performance, 31*(4), 749-761.

Brooks, R. A. (1991). Intelligence without representation. *Artificial Intelligence, 47*, 139-159.

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

Catrambone, R., & Yuasa, M. (2006). Acquisition of procedures: The effects of example elaborations and active learning exercises. *Learning and Instruction, 6*, 139-153.

Chi, M. T. H. (2006). Two approaches to the study of experts' characteristics. In K. A. Ericsson, N. Charness, P. J. Feltovich & R. R. Hoffman (Eds.), *The Cambridge handbook of expertise and expert performance* (pp. 21-30). New York: Cambridge University Press.

Clark, A. (1997). *Being There: Putting Brain, Body and World together again.* Cambridge, MA: MIT Press.

Fennell, K., Sherry, L., Roberts, R. J., & Feary, M. (2006). Difficult access: the impact of recall steps on flight management system errors. *International Journal of Aviation Psychology, 16*(2), 175-196.

Fikes, R. E., & Nilsson, N. J. (1971). STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, *2*, 189-208.

Hazeltine, E., Teague, D., & Ivry, R. B. (2002). Simultaneous dual-task performance reveals parallel response selection after practice. *Journal of Experimental Psychology: Human Perception and Performance, 28*, 527-545.

Hutchins, E. (1995). How a Cockpit remembers its Speeds. *Cognitive Science, 19*, 265-288.

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. Mondell & J. Driver (Eds.), *Control of cognitive processes: Attention and performance XVIII* (pp. 681-712). Cambridge, MA: MIT Press.

Larkin, J. H. (1989). Display-based problem solving. In D. Klahr & K. Kotovsky (Eds.), *Complex information processing: the impact of Herbert A. Simon* (pp. 319-341). Hillsdale, NJ: Erlbaum.

Lebiere, C., Wallach, D., & Taatgen, N. A. (1998). Implicit and explicit learning in ACT-R. In F. E. Ritter & R. M. Young (Eds.), *Second European conference on cognitive modelling* (pp. 183-189). Nottingham, UK: Nottingham university press.

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

McCarthy, J., & Hayes, P. J. (1969). Some philosophical problems from the standpoint of artificial intelligence. In B. Meltzer, D. Michie & M. Swann (Eds.), *Machine Intelligence* (Vol. 4, pp. 364-502). Edinburgh, Scotland: Edinburgh University Press.

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

Newell, A., & Rosenbloom, P. S. (1981). Mechanisms of skill acquisition and the law of practice. In J. R. Anderson (Ed.), *Cognitive skills and their acquisition* (pp. 1-56). Hillsdale, NJ: Erlbaum.

Newell, A., & Simon, H. (1963). GPS, a program that simulates human thought. In E. A. Feigenbaum & J. Feldman (Eds.), *Computers and Thought*. New York: McGraw-Hill.

Roberts, S., & Pashler, H. (2000). How persuasive is a good fit? A comment on theory testing. *Psychological Review, 107*(2), 358-367.

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

Sherry, L., Polson, P., Fennell, L., & Feary, M. (2002). *Drinking from the Fire Hose: Why the FMC/MCDU can be hard to learn and difficult to use.* Honeywell publication No. 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*(3), 421-455.

Taatgen, N. A. (in press). The minimal control principle. In W. Gray (Ed.), *Integrated models of cognitive systems*. Oxford: Oxford University Press.

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.

Author Note

Niels A. Taatgen, Psychology, Carnegie Mellon University and Artificial Intelligence, University of Groningen; David Huss and John R. Anderson, Psychology, Carnegie Mellon University.

This research was supported by NASA grant NRA2-38169. We thank Stefani Nellen and Ion Juvina for their helpful comments on the manuscript, Daniel Dickison and Ion Juvina for collecting the data, and Peter Polson and Karl Fennell for their advice.

Correspondence concerning this article should be addressed to Niels Taatgen, Department of Psychology, Carnegie Mellon University, Pittsburgh PA 15213. E-mail: taatgen@cmu.edu.

Table 1

Instructions in the list and in the context condition

List condition	Context condition
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	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, by pressing the line key next to it. 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.
Remove-discontinuity: 1. Press the LEGS key 2. Press the line select key after the discontinuity 3. Press the line key with the THEN prompt 4. Press EXEC	Discontinuities in the route When the text "Route discontinuity" appears on the LEGS page, the route should be made continuous. In order to make the route continuous, two points on the route have to be reconnected. The LEGS page will show the last connected waypoint followed by "THEN" and a line with boxes. Enter the waypoint that you want to fly to after the last connected waypoint on that line.

Table 2

Representation of the instructions in the model for the two conditions of the experiment.

List condition			Context condition		
Pre	Action	Post	Pre	Action	Post
Start	Press LEGS	Direct 1	On INIT page	Press LEGS	On LEGS page
Direct 1	Type Destination	Direct 2	On LEGS page	Type Destination	Destination in Scratchpad
Direct 1	Press line key with destination	Direct 2	On LEGS page	Press line key with destination	Destination in Scratchpad
Direct 2	Press 1L	Direct 3	Destination in Scratchpad	Press 1L	Modification done
Direct 3	Check for discontinuity	Direct 4 or Direct 6*	Modification done	Check Navigational Display	Check ok
Direct 4	Check Navigational Display	Direct 5	Check ok	Press EXEC	Goal Achieved
Direct 5	Press EXEC	Goal Achieved			
Discon 6	Press LEGS	Discon 7	Discontinuity	Press line key after discontinuity	Discontinuity in Scratchpad
Discon 7	Press line key after discontinuity	Discon 8	Discontinuity in Scratchpad	Press line key with discontinuity	Modification done
Discon 8	Press line key with discontinuity	Discon 9			
Discon 9	Press EXEC	Direct 4			

* This step sets the expected state to Direct 4 when there is no discontinuity, or to Direct 6 when there is one.

Table 3

Example of production compilation

Original rule 1:	Original rule 2:	Learned rule:
IF the goal is to do a task	IF the goal is to do a task	IF the goal is to do the FMS task
THEN send a request to declarative memory for an operator for that task	AND an operator that specifies a press action has been retrieved from declarative memory	AND the state is Not on the LEGS page
		THEN Press the LEGS key
Operator-example:	AND the precondition of the operator matches the current state	AND set the expected state to "on LEGS page"
Precondition Not on LEGS page	THEN Press the specified key	
Action Press LEGS key	AND set the expected state to the postcondition of the operator	
Postcondition On LEGS page		

Table 4

Proportions correct in Experiment 2 by problem type. t-tests are Welch one-sided.

Problem type	List correct	Context correct	Significance	Model List correct	Model Context correct
Easy	92.2%	91.7%	$t<1$	90.6%	98.2%
Medium	77.2%	92.8%	$t(32.0)=2.28$, $p=0.015$	83.3%	96.8%
Hard	76.7%	91.1%	$t(26.6)=2.47$, $p=0.010$	84.9%	92.6%
Copilot without error	86.9%	95%	$t(22.8)=1.47$, $p=0.077$	86.5%	96.4%
Copilot with error	66.7%	77.8%	$t(24.9)=1.86$, $p=0.037$	67.4%	69.2%

Footnote

1 The model of the experiment is available for downloading from the ACT-R models page: http://act-r.psy.cmu.edu/models.

Figure Captions

Figure 1. The FMS experiment. The left of the display shows the keyboard and displays 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". The rectangles left and right of the FMS display are also keys, called line keys. The keys to the left of the display are names 1L to 6L, and the keys to right 1R to 6R. The bottom line on the FMS display is called the scratchpad (which is empty in the figure). Text typed on the keyboard will appear in the scratchpad, and can be transferred to a line in the display by pushing a line key. For example, typing "EMA" puts EMA in the scratchpad. Pushing the 1L key would then put EMA next to the 1L key on the display, replacing ALICE. An alternative method to put text in the scratchpad is by pushing one of the line keys. For example, pushing the 1L key copies ALICE into the scratchpad. The LEGS page, which is currently displayed, lists the current route the plane is taking, which starts with ALICE-BOYDD-CONEE-BAETT-AYMAN, and continues on two more pages. The Navigation Display on the top-right shows a graphical version of the route. The triangle represents the current position of the plane with a line connecting the upcoming waypoints.

Figure 2. Simplified display of the direct-to procedure based on Figure 1. This example assumes Air Traffic Control has given the directive to proceed directly to BAETT and that the FMS is initially not on the LEGS page. Pressing the LEGS key brings it to the LEGS page, showing BAETT as the fourth waypoint in the flight plan. Pressing the 4L key, which is next to BAETT, brings it to the scratchpad (the bottom line on the display). Pressing 1L will put it in the top line of the flight plan as a modification (indicated by white-on-black). The next step is to verify that

the modification is what was intended. The navigational display (ND) indeed shows a dotted line from the airplane (the triangle) directly to BAETT. Finally the EXEC key is pushed making the modification final.

Figure 3. Proportion correct for all the 36 problems in the two conditions, which are divided in 3 blocks with 3 levels of difficulty each. Circles are the empirical data with standard error bars, and the dotted line with crosses is the model fit that will be discussed later.

Figure 4. Average solution times for correctly solved problems for all 36 problems. Circles are the empirical data with standard error bars, and the dotted line with crosses is the model fit that will be discussed later.

Figure 5. Overview of the ACT-R Architecture

Figure 6. Outline of the model

Figure 7. Frequency of presses on the LEGS key for the three blocks and two conditions, empirical data and model

Figure 8. Frequency of presses on the EXEC key for the three blocks and two conditions, empirical data and model, and whether or not the problem involved a discontinuity (which according to the literal list instructions would require a second press on EXEC)

Figure 9. Proportion correct for all the 72 problems in the two conditions averaged in groups of three. The experiment is divided in 3 blocks with 5 types of problems each (e=easy, m=medium, h=hard, co=copilot, wco=copilot with error). Circles are the empirical data with standard error bars, and the dashed line represents the model prediction. Although in the real experiment co and wco problems were mixed, they are sorted into separate categories within a block for clarity.

Figure 10. Average solution times for correctly solved problems for all 72 problems averaged in groups of three. The experiment is divided in 3 blocks with 5 types of problems each (e=easy,

m=medium, h=hard, co=copilot, wco=copilot with error). Circles are the empirical data with standard error bars, and the dashed line represents the model prediction.

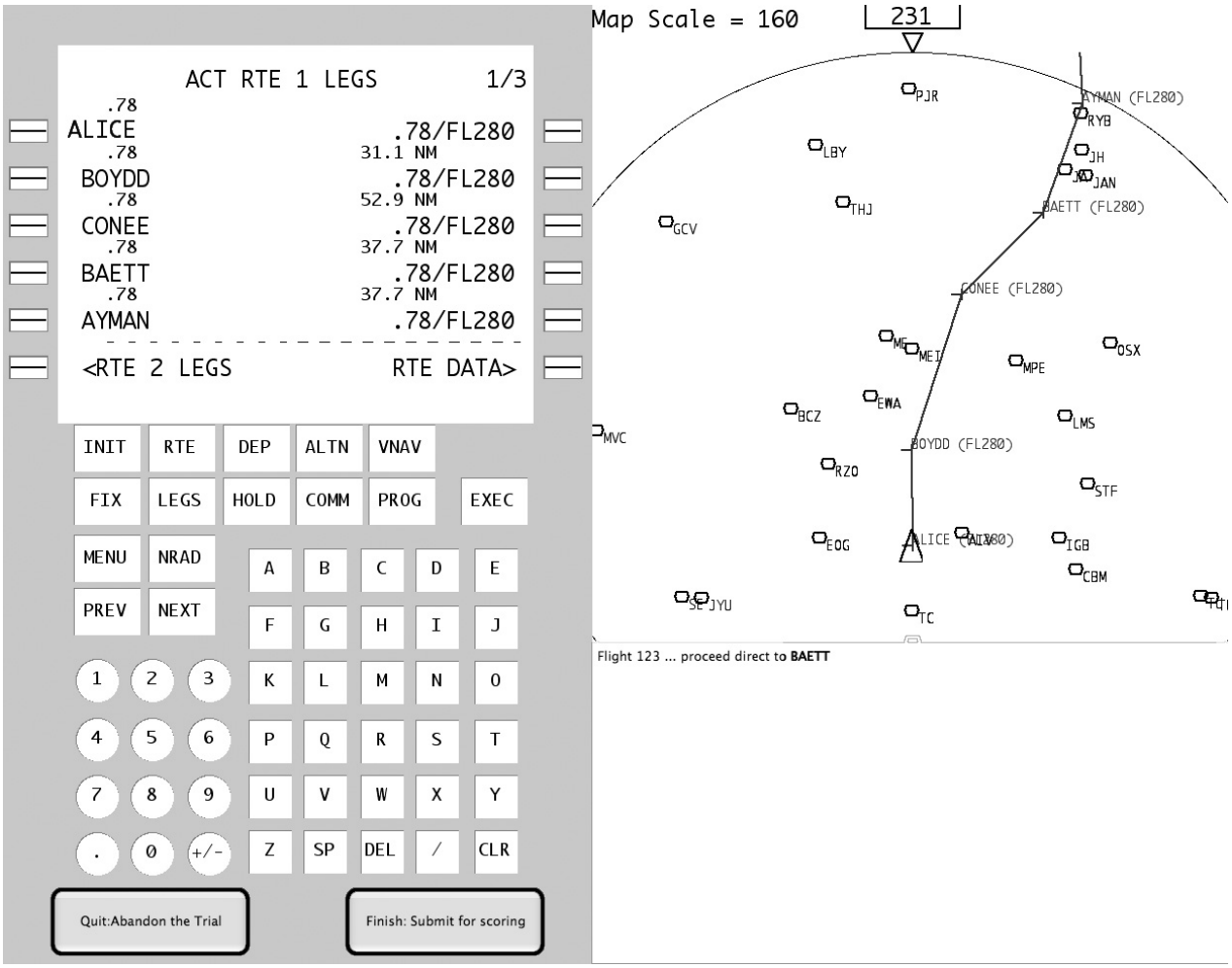

ACT RTE 1 LEGS 1/3
.78
ALICE .78/FL280
.78 31.1 NM
BOYDD .78/FL280
.78 52.9 NM
CONEE .78/FL280
.78 37.7 NM
BAETT .78/FL280
.78 37.7 NM
AYMAN .78/FL280
<RTE 2 LEGS RTE DATA>
INIT RTE DEP ALTN VNAV
FIX LEGS HOLD COMM PROG EXEC
MENU NRAD
PREV NEXT
A B C D E
F G H I J
K L M N O
P Q R S T
U V W X Y
Z SP DEL / CLR
1 2 3
4 5 6
7 8 9
. 0 +/-
Quit:Abandon the Trial
Finish: Submit for scoring
Map Scale = 160
231
AYMAN (FL280)
BAETT (FL280)
CONEE (FL280)
BOYDD (FL280)
ALICE (FL280)
PJR
RYB
LBY
JH
JAN
THJ
GCV
OSX
MPE
EWA
BCZ
LMS
MVC
RZO
STF
EOG
IGB
CBM
JYU
TC
Flight 123 ... proceed direct to BAETT

Figure 1

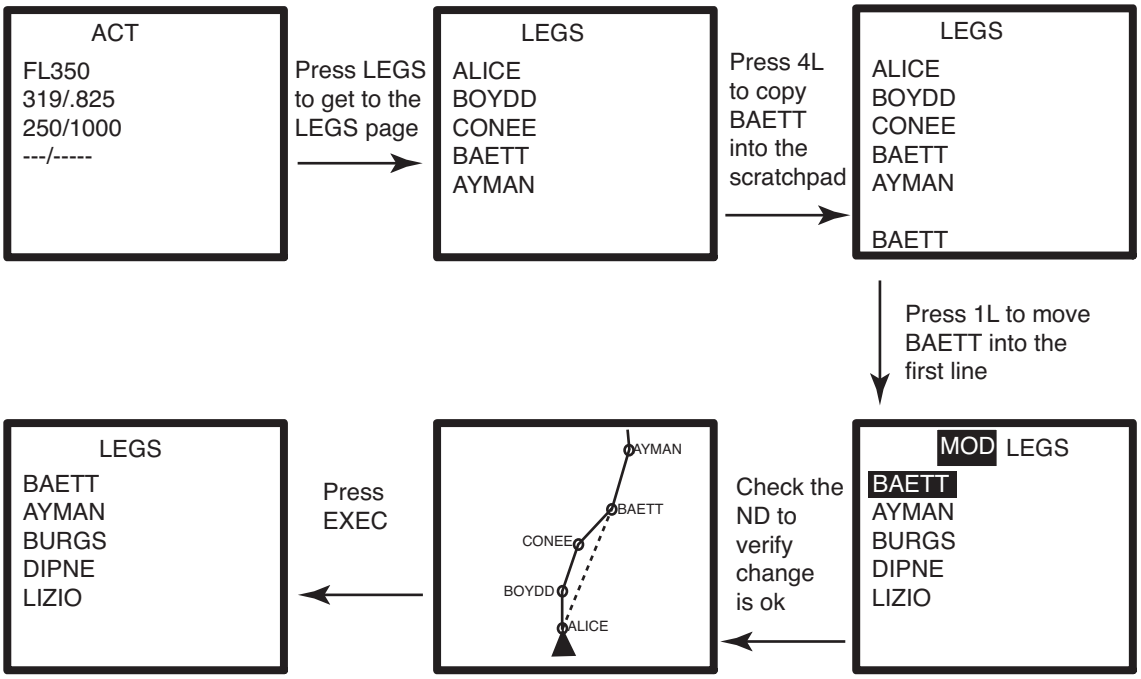

ACT
FL350
319/.825
250/1000
---/-----
Press LEGS to get to the LEGS page
LEGS
ALICE
BOYDD
CONEE
BAETT
AYMAN
Press 4L to copy BAETT into the scratchpad
LEGS
ALICE
BOYDD
CONEE
BAETT
AYMAN
BAETT
Press 1L to move BAETT into the first line
MOD LEGS
BAETT
AYMAN
BURGS
DIPNE
LIZIO
Check the ND to verify change is ok
AYMAN
BAETT
CONEE
BOYDD
ALICE
Press EXEC
LEGS
BAETT
AYMAN
BURGS
DIPNE
LIZIO

Figure 2

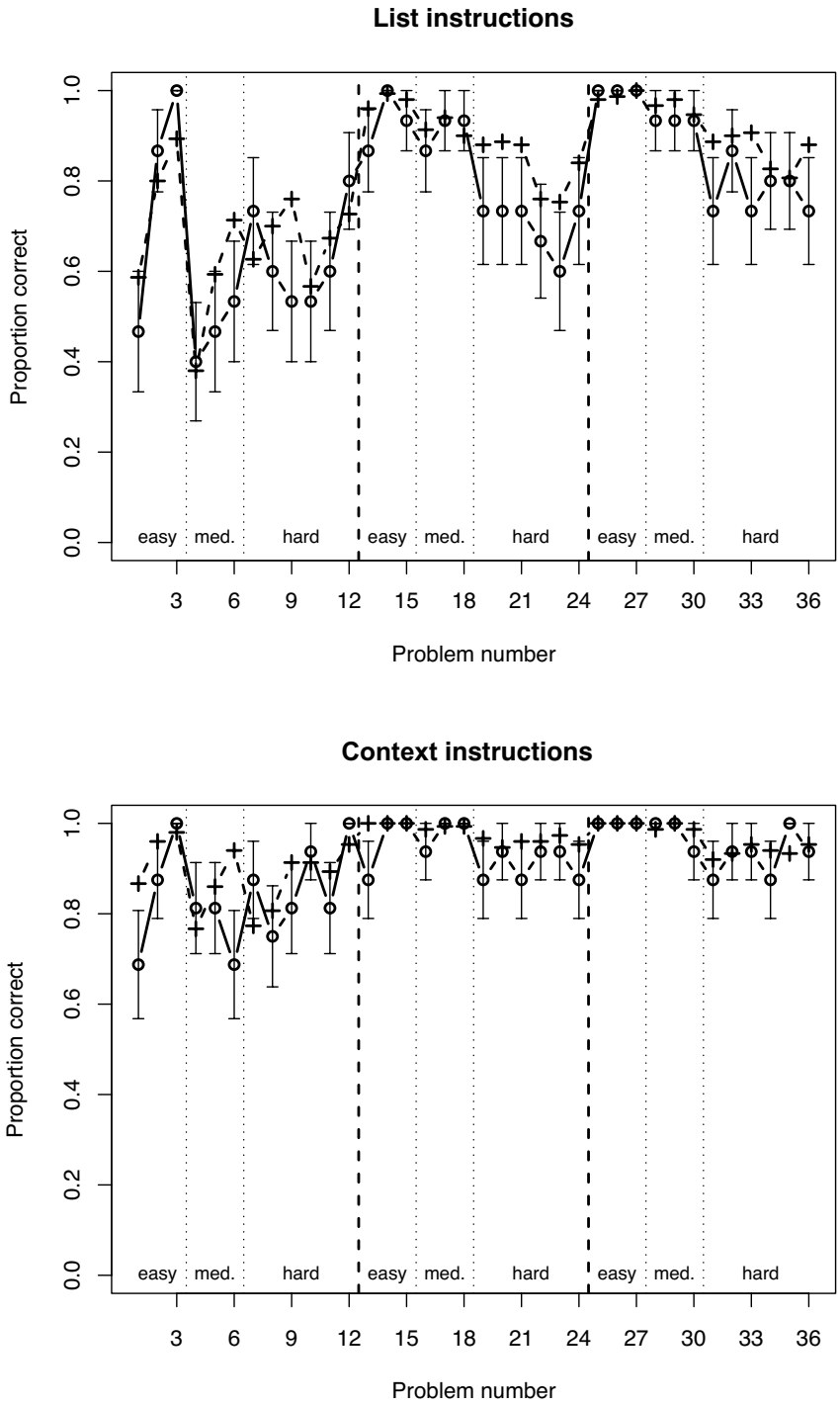

List instructions
Proportion correct
0.0
0.2
0.4
0.6
0.8
1.0
easy
med.
hard
easy
med.
hard
easy
med.
hard
3
6
9
12
15
18
21
24
27
30
33
36
Problem number
Context instructions
Proportion correct
0.0
0.2
0.4
0.6
0.8
1.0
easy
med.
hard
easy
med.
hard
easy
med.
hard
3
6
9
12
15
18
21
24
27
30
33
36
Problem number

Figure 3

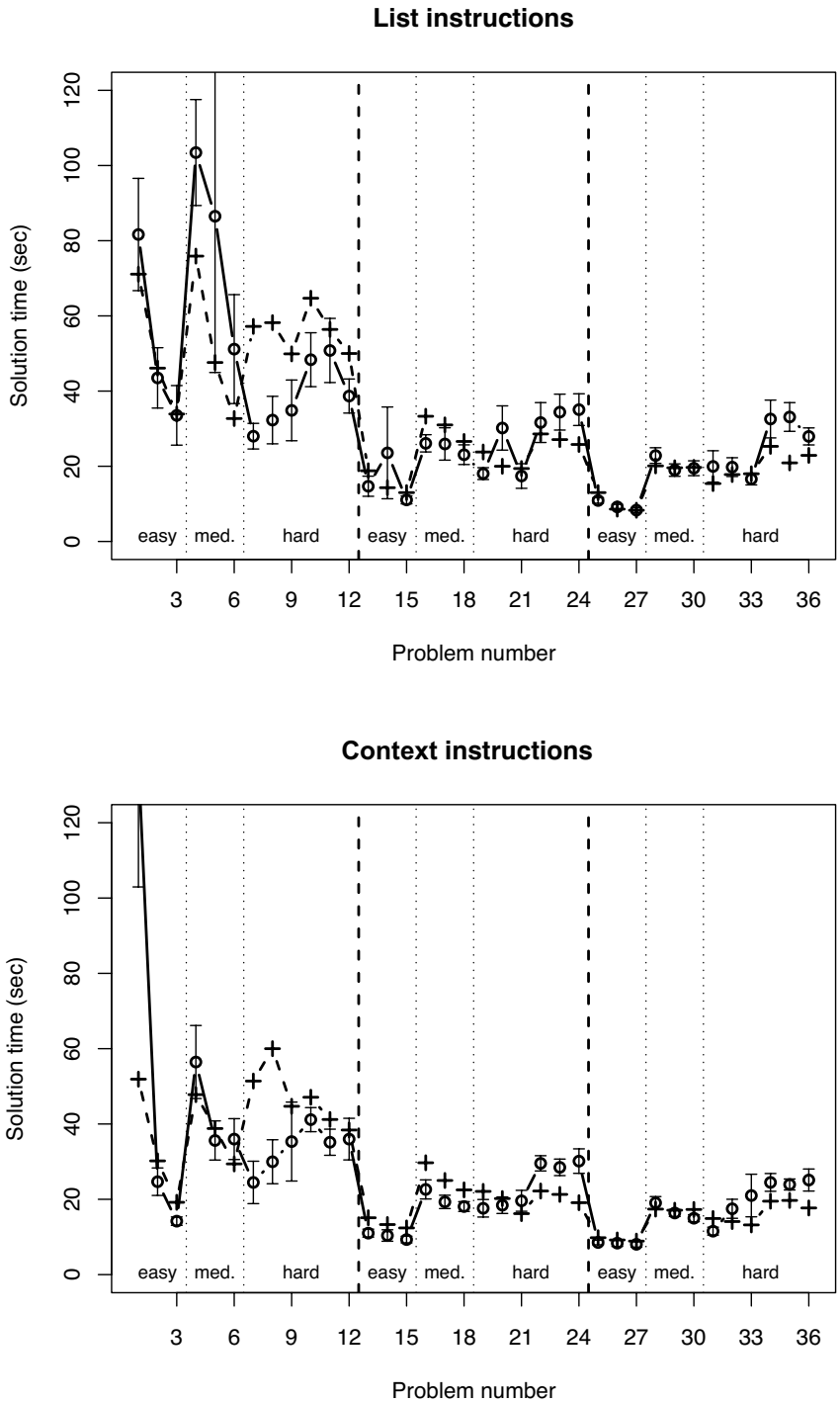
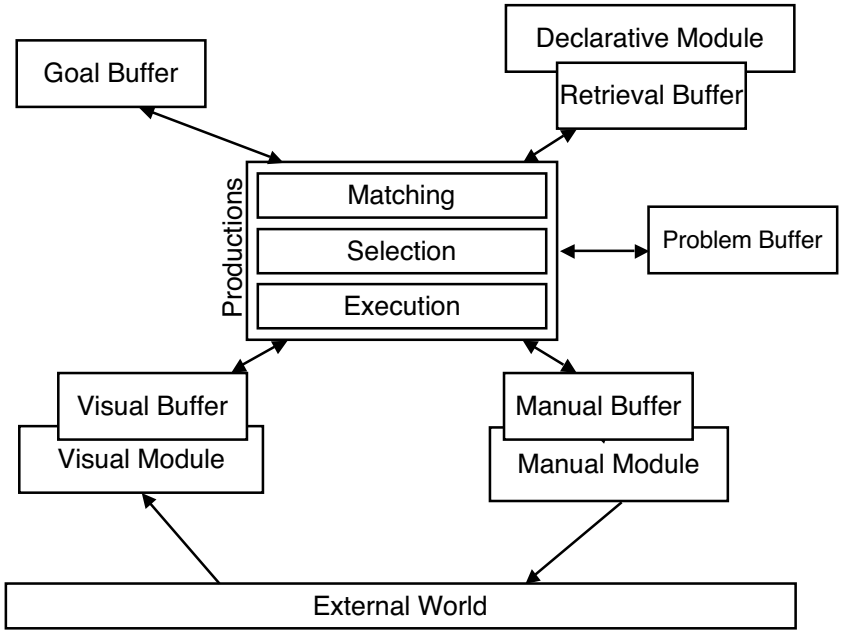

List instructions
Solution time (sec)
0
20
40
60
80
100
120
easy
med.
hard
easy
med.
hard
easy
med.
hard
3
6
9
12
15
18
21
24
27
30
33
36
Problem number
Context instructions
Solution time (sec)
0
20
40
60
80
100
120
easy
med.
hard
easy
med.
hard
easy
med.
hard
3
6
9
12
15
18
21
24
27
30
33
36
Problem number

Figure 4

Figure 5


Declarative Module
Goal Buffer
Retrieval Buffer
Productions
Matching
Selection
Execution
Problem Buffer
Visual Buffer
Visual Module
Manual Buffer
Manual Module
External World

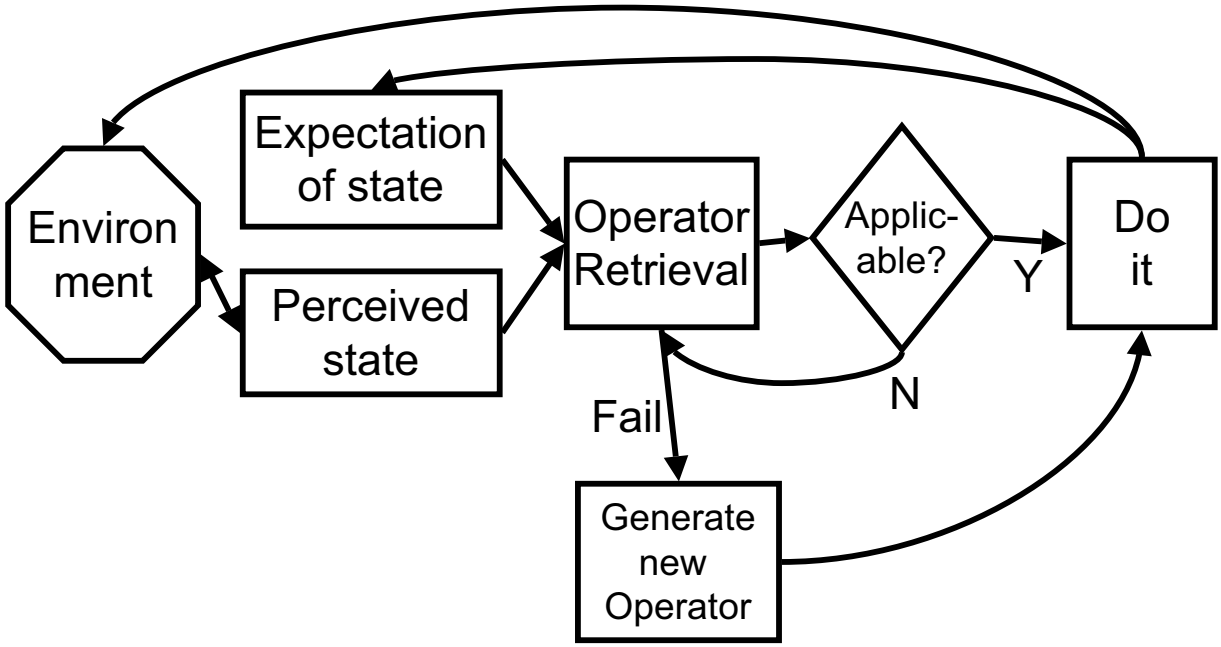

Environ
ment
Expectation
of state
Perceived
state
Operator
Retrieval
Applic-
able?
Y
N
Do
it
Fail
Generate
new
Operator

Figure 6.

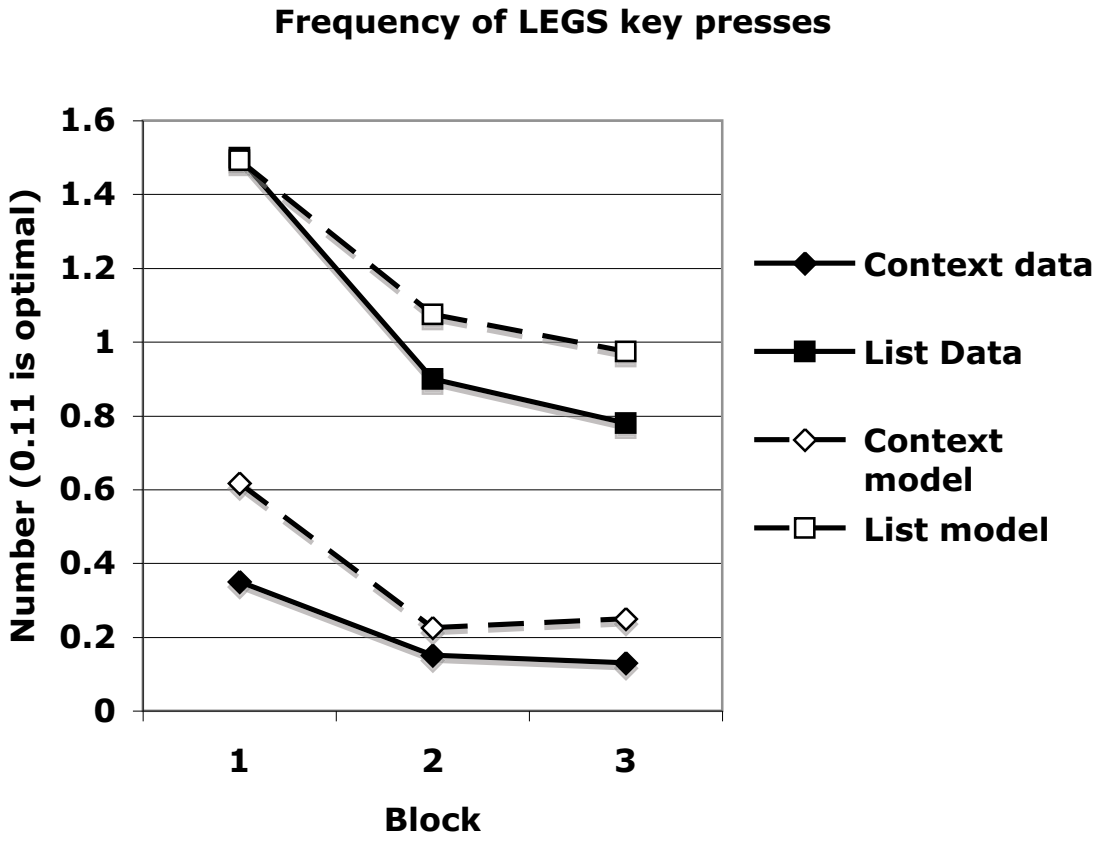

Frequency of LEGS key presses
Number (0.11 is optimal)
1.6
1.4
1.2
1
0.8
0.6
0.4
0.2
0
1
2
3
Block
Context data
List Data
Context model
List model

Figure 7.

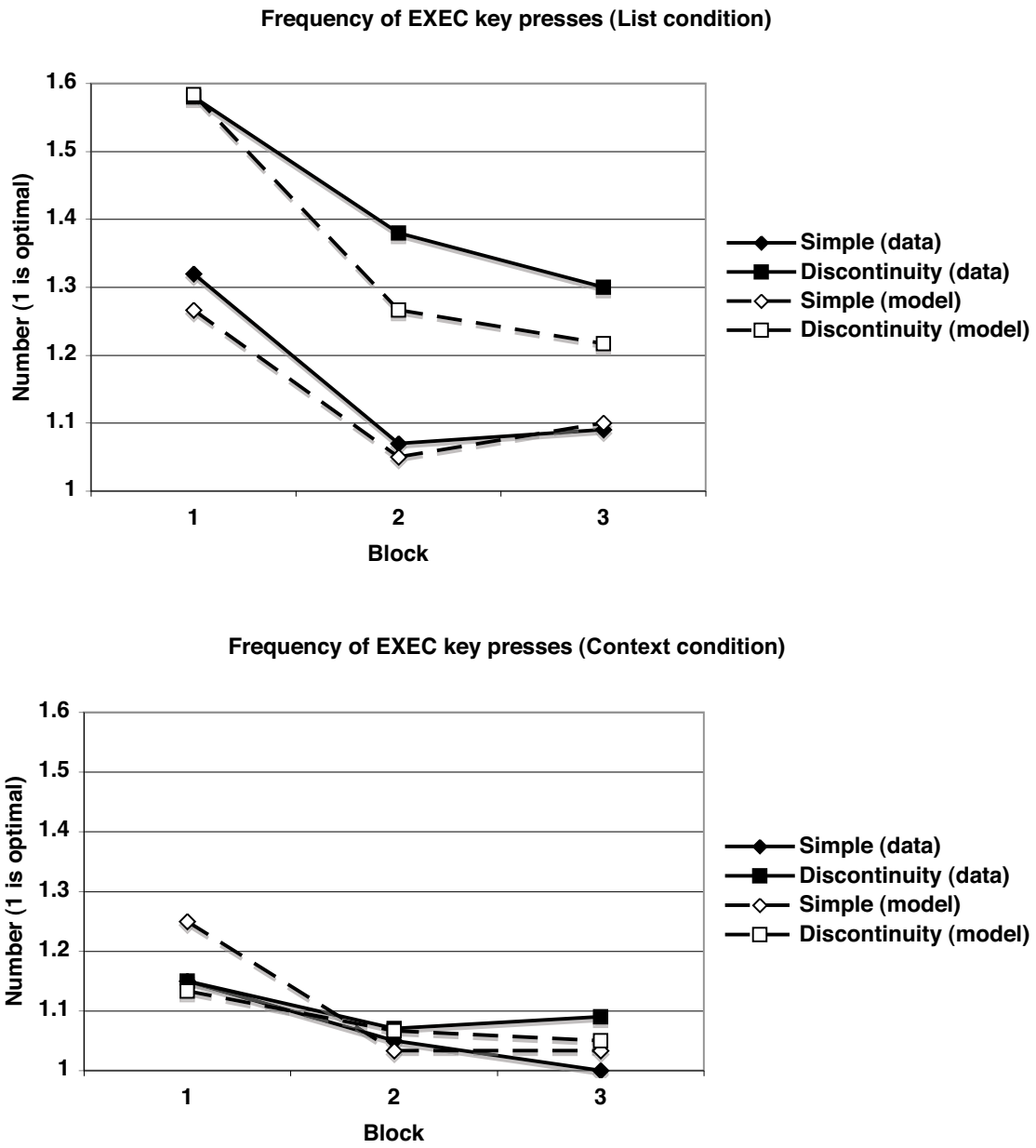

Frequency of EXEC key presses (List condition)
Number (1 is optimal)
1.6
1.5
1.4
1.3
1.2
1.1
1
1
2
3
Block
Simple (data)
Discontinuity (data)
Simple (model)
Discontinuity (model)
Frequency of EXEC key presses (Context condition)
Number (1 is optimal)
1.6
1.5
1.4
1.3
1.2
1.1
1
1
2
3
Block
Simple (data)
Discontinuity (data)
Simple (model)
Discontinuity (model)

Figure 8.

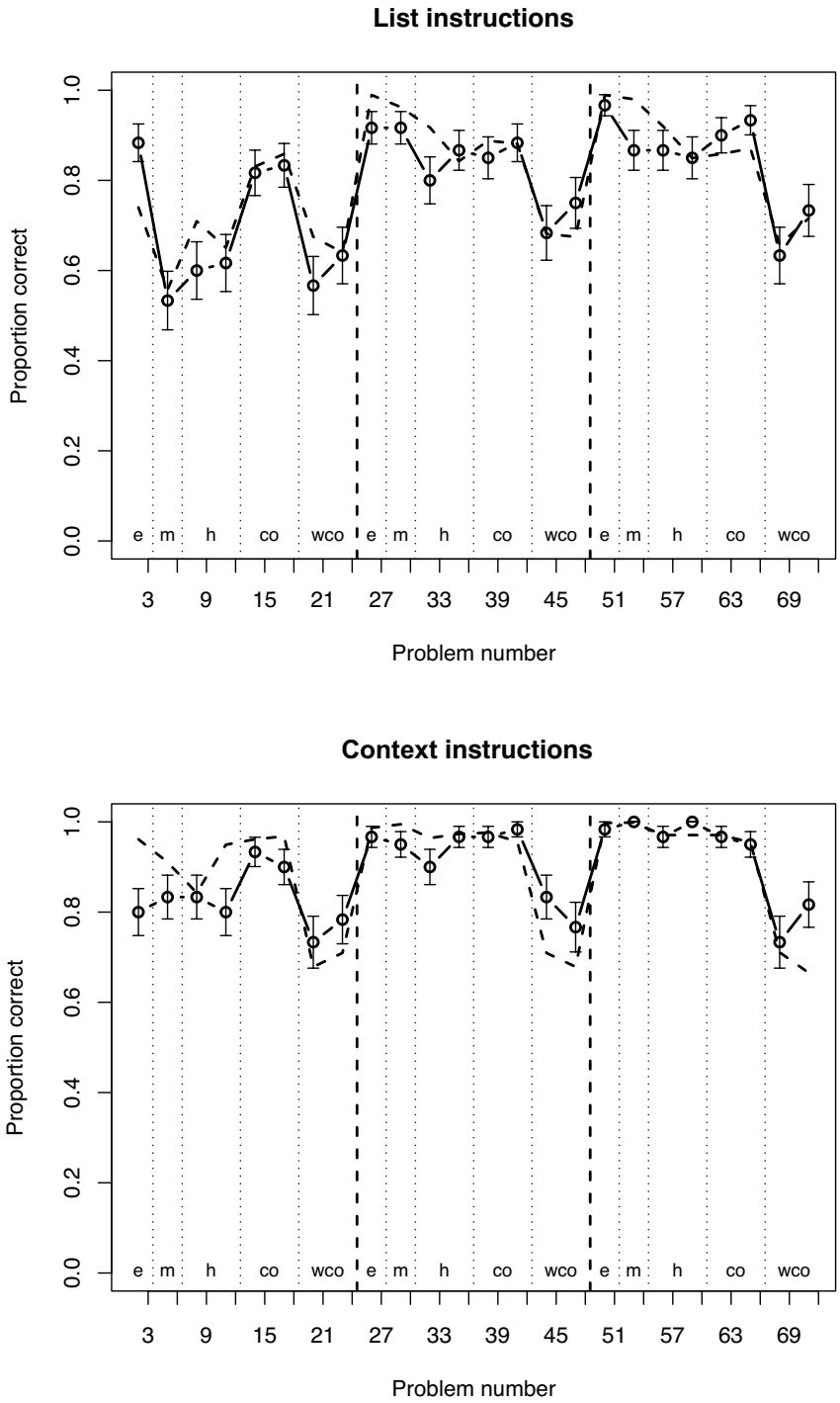

List instructions
Proportion correct
0.0
0.2
0.4
0.6
0.8
1.0
e m h co wco e m h co wco e m h co wco
3 9 15 21 27 33 39 45 51 57 63 69
Problem number
Context instructions
Proportion correct
0.0
0.2
0.4
0.6
0.8
1.0
e m h co wco e m h co wco e m h co wco
3 9 15 21 27 33 39 45 51 57 63 69
Problem number

Figure 9.

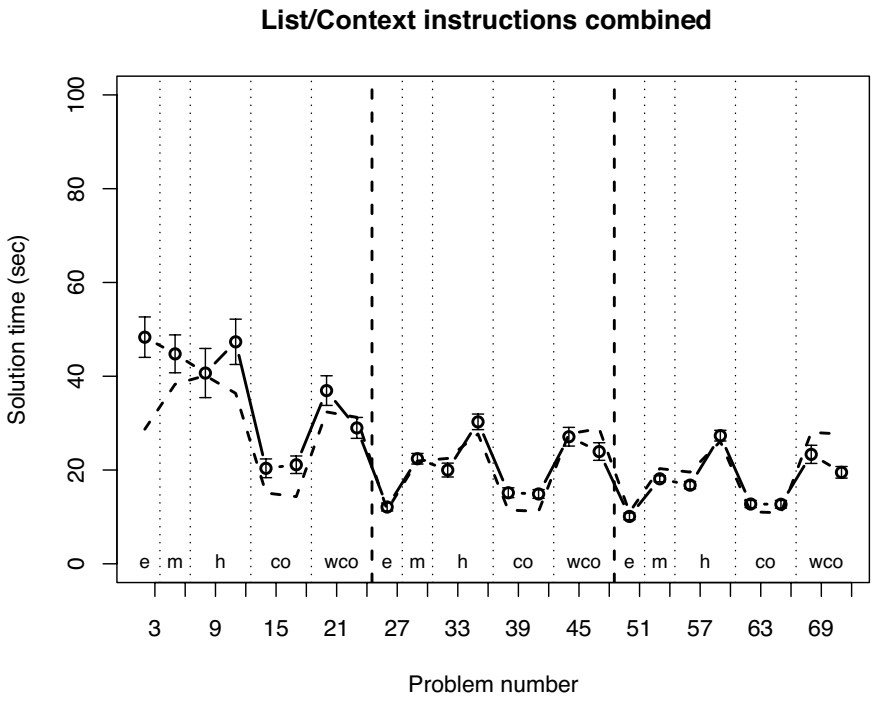

List/Context instructions combined
Solution time (sec)
0
20
40
60
80
100
e
m
h
co
wco
e
m
h
co
wco
e
m
h
co
wco
3
9
15
21
27
33
39
45
51
57
63
69
Problem number

Figure 10