Ritter, F. E., & Bibby, P. A. (2008). Modeling how, when, and what learning happens in a diagrammatic reasoning task. Cognitive Science, 32, 862-892.

MODELING HOW, WHEN, AND WHAT IS LEARNED IN A SIMPLE FAULT-FINDING TASK

Frank E. Ritter
College of Information Sciences and Technology
Penn State
University Park, PA 16802
and
Peter A. Bibby
School of Psychology
University of Nottingham
Nottingham, England NG7 2RD

+1 814 865-4453 (ph)
frank.ritter@psu.edu

Abstract

We have developed a process model that learns in multiple ways while finding faults in a simple control panel device. The model predicts human participants' learning through its own learning. The model's performance was systematically compared to human learning data, including the time course and specific sequence of learned behaviors. These comparisons show that the model accounts very well for measures such as problem-solving strategy, the relative difficulty of faults, and average fault-finding time. More importantly, because the model learns and transfers its learning across problems, it also accounts for the faster problem-solving times due to learning when examined across participants, across faults, and across the series of 20 trials on an individual participant basis. The model shows how learning while problem solving can lead to more recognition-based performance, and helps explain how the shape of the learning curve can arise through learning and be modified by differential transfer. Overall, the quality of the correspondence appears to have arisen from procedural, declarative, and episodic learning all taking place within individual problem solving episodes.

Keywords: Artificial Intelligence, Psychology; Skill acquisition and learning, Problem Solving, Cognitive architecture, Soar, Human-computer interaction; Symbolic computational modeling, Human experimentation.

1. INTRODUCTION

There is a speedup in the time taken to solve problems that is nearly universal across tasks and participants (Rosenbloom & Newell, 1987). A major question has been understanding how this speedup occurs. Cognitive architectures and models have proposed several mechanisms to account for this learning, such as procedural knowledge compilation (e.g., Anderson, 2007; Gonzalez, Lerch, & Lebiere, 2003; Pavlik, 2007; Taatgen & Lee, 2003) and base level learning (Altmann & Trafton, 2002) in ACT-R, connection strengthening in PDP models (e.g., O'Reilly & Munakata, 2000), rule creation from impasses using analogical reasoning (VanLehn & Jones, 1993), constraint elaboration (Ohlsson, 2007), and both declarative and procedural learning mechanisms (Feigenbaum & Simon, 1984; Gobet & Lane, 2007; Larkin, 1981; Rosenbloom & Newell, 1987).

Each of these learning mechanisms has successfully accounted for some aspects of this speedup, and as a group they have proven useful in explaining patterns of learning in a variety of domains (e.g., the Tower of Hanoi: Altmann & Trafton, 2002; Anzai & Simon, 1979; Ruiz & Newell, 1989; physics problem solving: Larkin, 1981; Lisp programming, text editing, and simple differential calculus: Singley & Anderson, 1989; computer games: Bauer & John, 1995; and complex interface use: Gonzalez, Lerch, & Lebiere, 2003; Lee & Anderson, 2001; Taatgen & Lee, 2003). Problematically, the learning produced by these mechanisms has been compared with aggregate data. Response times aggregated across trials, participants, or both, have been used most often to test the models (e.g., Ram, Narayanan, & Cox, 1995; Taatgen & Lee, 2003)[1].

The method of comparison between models and behavior has varied. A few models have been compared with behavior before and after extensive amounts of practice performing a task. That is, the models using these mechanisms started by performing the task like a novice participant, and after extensive application of the learning mechanisms performed like expert participants (e.g., Larkin, 1981), and recent models have matched learning at several points in time (e.g., Anderson, Bothel, Byrne, Douglass, Lebiere, & Qin, 2004; Gonzalez et al., 2003; Taatgen & Lee, 2003).

There are a number of issues that remain in understanding how problem solving is learned. First is the actual path of learning by an individual. It takes several years for a person to become expert in problem solving in a complex domain. When comparing novices and experts in such domains, the cross-sectional methodology used prevents acquiring a full understanding of the learning process as it occurs. For example, Able (Larkin, 1981) was a model of the transition

between novice and expert behavior in physics problem solving in terms of the application order of domain principles. It used an impasse-driven learning mechanism to learn rules automatically while solving problems. The acquisition of these rules represented the transition from novice to expert. However, only the model's initial and final behavior were compared with novice and expert behavior. Modeling exercises of this kind inevitably hide a large proportion of the learning that takes place. At the same time, models of this kind do provide useful insights into the learning process.

Second, there is the issue of the grain size of the measures analyzed. Most models have been compared with aggregate reaction time or performance data. Only a few models have examined the details of problem-solving strategy. For example, Anzai and Simon (1979) observed the changes in strategy across four trials of solving the Tower of Hanoi and modeled these changes using an adaptive production system. The model's behavior was not compared to the participant's behavior at an action-by-action level, but Anzai and Simon did show that the model produced the same qualitative strategy shifts as the participant. Other researchers have focused on the goals that participants have attempted to reach as they solve problems. Cascade (R. M. Jones & VanLehn, 1992; VanLehn & Jones, 1993) modeled how good and poor students learn while reading solved physics problems. In its most complete comparison, Cascade was used to model nine participants' behavior individually while studying 28 worked physics problems, learning by examining the problems and sometimes explaining the steps to themselves. Cascade predicted the goals that participants would explain and how what was learned would transfer to other problems. This work did not examine the time course of learning, but it is otherwise a very good example of where the detailed predictions of a model that learns have been verified with behavioral data.

Third, there is the issue of the automatic learning mechanism. In detailed studies, VanLehn (1989, 1991) examined learning in several domains including the Tower of Hanoi. He modeled learning by proposing when rules were learned and what the learned rules contained, but these rules were created by hand. There are several ACT-R models that also come close to examining how and when learning occurs. While the ACT architecture implementations include learning capabilities, models built within this framework often have modeled learning by adding hand-written rules based on the participant's performance (e.g., the ACT-R tutors, Anderson, Conrad, & Corbett, 1989). There are also models that learned automatically in both ACT-R (Gonzalez et al., 2003; G. Jones, Ritter, & Wood, 2000; Pavlik, 2007; Taatgen, 2005; Taatgen & Lee, 2003) and Soar (Altmann & John, 1999; Altmann, Larkin, & John, 1995; Bauer & John, 1995) and that have been compared with human problem-solving protocols but these comparisons did not examine the time course of individuals' learning, or did not match individuals very well (Nerb,

Ritter, & Krems, 1999).

Is it possible to construct a general learning mechanism that could capture learning at the rate learning occurs in individuals? Many would believe the answer to this question is yes, but a general learning mechanism has yet to be implemented and compared with the time-course of individual behavior.

While prior work on learning within cognitive models has advanced the scientific understanding of learning in specific domains and across participants, these issues (individual learning paths, time granularity, and automatic learning mechanism) are the next issues in understanding cognitive skill development. This paper reports the results of comparing the performance of a process model that learns to participants' behavior. It addresses these issues by:

(a) comparing participants' behavior individually with the model's behavior as they both complete 20 problem-solving tasks.
(b) comparing participants' behavior with the model's behavior using several different kinds of aggregation.
(c) using a computational architecture that has an automatic learning mechanism, thus avoiding the need to hand-craft production rules that capture learning.

As a result, the method, model, and analyses proposed in this paper constitutes a new theory of learning and performance improvement that accounts for individual differences in learning at a low time granularity via a consistent (but flexible and generalizable) learning mechanism. From this, we can draw new scientific insight into the nature of the famous (but in completely understood) learning curve at a far deeper level than was possible in prior work.

We begin by explaining the task. Next, we explain the model and how we gathered the human data. Next, we offer a detailed comparison between *both* aggregate and individual measures of the model's and participants' performance, which provides empirical support for the model. We conclude by examining the implications of this work for problem solving, including how the different types of learning contribute to this result, the learning curve, and reasoning with diagrams.

Because we believe that science should proceed in the most transparent fashion possible, we have put the model and its architecture, a reimplementation of the stimuli and instructions, and the data and most of the analyses on the web (acs.ist.psu.edu/projects/diag/). This may help set a standard for models in this area (cf. Thimbleby, 2004), and will help other scientists to evaluate, extend, and improve upon our work.

1.1 The fault-finding task and data collection methodology

Participants and the model solved the same task, trouble-shooting a control panel device using a memorized schematic. The device's interface, its internal schematic, and task instructions are shown in Fig. 1. The device is similar to Kieras and Bovair's (1984) laser-bank device. Prior research using a similar task (Kieras, 1988; Kieras & Bovair, 1984) has shown that instructions for this task need to convey to participants knowledge of the structure of the system, knowledge about the underlying principles that control the behavior of the system, and knowledge about how to perform tasks using the structure and principle information.

Ten participants, all University of Nottingham undergraduates aged between 19 and 21 years were paid £5 for participating. At the beginning of each data collection session participants were given a picture of the control panel (an example is shown in Fig. 1a), a general introduction to the device (Fig. 1b), and a schematic of the underlying circuit (see Fig. 1c) showing how the components of the system were connected through a series of switches. Participants were asked to memorize these materials. While studying the instructions they were allowed to ask questions of the experimenter who fully understood the task. The participants were also told that a fault existed when a component indicator light was not lit when it was receiving power. Taken together, this information was sufficient for participants to be able to identify faults.

Participants were asked to study the instructional materials, paying particular attention to the schematic representing the organization of the components and switches. After a period of 5 minutes participants were asked to draw the schematic. If they could accurately reproduce the schematic, the instructions were removed and the experiment began. If participants were unable to draw the schematic they were given an additional 5 minute period in which to study the instructional materials. No participant took more than 10 minutes to learn the materials.

< Insert Fig. 1 about here.>

Next, the fault-finding task was described in detail to the participants and a diagrammatic example of the task was given. Then participants were introduced to the device and shown how to select the faulty component. The device was simulated in Hypercard™ on an Apple™ Power Mac 4400. Participants were told that one component in the circuit was faulty and were asked to indicate it by clicking on it. Each participant received 20 example faults that they had to identify. This series was chosen randomly without replacement from a fixed set of 20 faults, and thus the series varied for each participant[2]. Participants' reaction times and choices were recorded and analyzed for latency and number of correct choices.

1.2 The Diag model

We constructed a computational process model in the Soar cognitive architecture that solves the fault-finding task. The model, called Diag, solves problems via search in solution spaces, and learns while doing the task. We will address each of these aspects (Soar, fault-finding, and learning in this section[3]).

1.2.1. The Soar architecture

The Diag model was built using the Soar 6 cognitive architecture (Laird, Newell, & Rosenbloom, 1987; Newell, 1990). Here we briefly describe the relevant aspects of the architecture that influence and support Diag's problem solving and learning functions. We refer the interested reader to more in-depth introductions to and explanations of the architecture (e.g., Lehman, Laird, & Rosenbloom, 1996; Lewis, 2001; Ritter, 2003), some of which are available online (Laird, 2003; Ritter & Young, 1996), including further information, references, and a more complete bibliography in the Soar FAQ.

Soar represents problem solving as search in problem spaces (Newell, 1980; Newell, Yost, Laird, Rosenbloom, & Altmann, 1991) and search through problem spaces (Ritter & Larkin, 1994). Operators transform states and are implemented with production rules that create and propose operators, and then implement operators after the architecture has selected an operator based on the proposals. Rules can also modify states directly, typically representing simple inferences, such as if a switch is on then power flows through it.

When there is a lack of knowledge about how to proceed, an impasse is declared. An impasse can be caused by, among other things, a lack of operators to apply, an operator that does not lead to changes in the information available for problem solving, or not knowing which of a set of operators to choose. An impasse is a data structure, similar to a goal, that allows further knowledge about how to resolve the impasse to be obtained through problem solving based on the particular impasse, including the state and operator (if one exists) that existed when the impasse occurred. Thus, the impasse is typically resolved by proposing an operator to apply to a state that did not have an operator, changing the state based on the operator that was being applied, or providing information about which operator to choose where there were multiple operators.

Soar models that learn have a hierarchically organized set of problem spaces, as problem solving in one impasse may lead to further impasses that are sometimes resolved in further problem spaces. When knowledge about how to resolve an impasse becomes available from

problem solving, Soar's learning mechanism[4] creates a new production rule. This new production rule will contain as its condition (left-hand side) the information in the higher context that led to the impasse so that the program can recognize the same situation in the future; it will contain as its action (right-hand side) the information that was acquired during the problem solving to resolve the impasse. In the future, the rule can eliminate the need for repeating the same problem solving when a similar situation (as defined by the condition) occurs. The bottom-up approach is where rules are learned from the bottom of a trace through the hierarchy. Soar was designed to model human learning rates (Rosenbloom & Newell, 1987), so we use it here.

1.2.2. How the model finds faults

The model begins at the level of minimum competence—the point at which participants are able to pass the training test showing that they have memorized the schematic, but before any significant gains in efficiency have been made. The model therefore assumes an understanding of power flow through the device, how to recognize a faulty component, and a basic knowledge about diagnosing a fault. The knowledge is available but has not been integrated.

Diag finds the fault using the circuit structure, working from left to right. Knowledge of the schematic is represented within the model as routes through the circuit as if the schematic diagram was memorized (thus the name, Diagrammatic, or Diag-Soar). Portions of these routes (such as EB1 is connected to MA) are recalled and followed to find the next component to examine. Interface information obtained from a simple model of vision about the states of the lights and switches is represented in declarative structures made up of attribute-value pairs. The combination of schematic and interface knowledge corresponds to the diagram experimental condition reported in Bibby and Payne (1993) and implements a common strategy, that of following paths through the circuit (Strategy 1 in Bibby & Payne, 1996).

Fig. 2 shows how the 20 operators that perform this problem-solving task are grouped into 7 hierarchically organized problem spaces. This hierarchical structure and Soar's learning mechanism are what gives rise to the learning while problem solving. These problem spaces and operators are implemented by 170 production rules. The model does not explain how this knowledge is acquired, which remains a substantial and open problem. Few models start from nothing and progress from task instructions to expert behavior (although see Anderson, 2007, Ch. 5, and Taatgen & Lee, 2003, for recent work in this area). These knowledge structures would arise out of natural language processing, processing of graphical representations, and problem solving to generate the initial task description.

Fig. 3 describes the model's performance, showing the operators and their order as performed by the model in solving the EB1 fault for the first time. The operator applications are numbered for reference. This figure was inspired by similar graphs (John, Vera, & Newell, 1994; Lehman et al., 1996; Newell, 1990). This graph illustrates the cyclical behavior of the model choosing a component to check and then checking it. We will use this figure to explain the model's behavior and structure. The full model trace is available from the web site (e.g., in EB1-trace.txt, a trace of the model solving the Energy Booster 1 fault). The explanation here provides a high level description of the model's behavior.

< Insert Figures 2 and 3 about here. >

Choosing a component to examine. The SOLVE-PROBLEM operator [labeled 1, Fig. 3] sets up the problem state. After it does this, no rules fire, and no operator is proposed to apply. The architecture creates an impasse (see the description of Soar above, or the online materials to read more about impasses in Soar) in the Diagnose space, noting that there is no operator to apply. The model then starts its pattern of interaction by working to choose a component to check (ATTEND), and comprehending what was seen (COMPREHEND). These actions are performed by operators in Soar. This pattern may be seen in the repeated return to the ATTEND and COMPREHEND operators [labeled 10-11, 24-25, 34-35, 43-44] at the top of Fig. 3 and later in Fig. 4.

To resolve the impasse (which is to create an operator in the DIAGNOSE problem space) the model selects a problem space to resolve this problem of a lack of operator in the top problem space. This space is FIND-FAULT, and it selects an operator, CHOOSE-COMPONENT [2], that chooses a component to attend to first. Initially, CHOOSE-COMPONENT cannot be applied directly without learning—the knowledge of which component to choose is in yet another problem space. Another impasse results, and the next operator, DIAGRAM-SUGGESTION [3] attempts to suggest which component to attend to based on knowledge about the diagram. It, too, initially lacks the knowledge to be applied, and a further suboperator, DIAGRAM-CHOICE [5], helps suggest which component to attend to based on the model's diagram (schematic) knowledge and what previous components have been checked. This knowledge is kept in the Diag-select problem space as a set of paths through the circuit. DIAGRAM-CHOICE [5] selects the next component to attend to using the previous component checked (or none) and the paths through the circuit. With no previously checked components, the power source is selected.

In less formal terms, the model knows the structure of the diagram and the structure of the interface, but has not compiled this into knowledge about which component to check at each point in time. Because knowledge in Soar is organized around the operators, problem spaces, and

their hierarchical relationship to each other, knowledge about how to implement an operator is often put in a lower problem space by the modeler to represent the knowledge state of a participant who has just studied the task instructions but not yet put them into practice. Learning compiles this knowledge into a more efficient form that is used in the higher problem space where the operator is applied.

As the DIAGRAM-CHOICE operator returns its result to the higher problem space, a rule is learned (LR-1, see Table 1)[5], that notes that when the operator DIAG-SUGGESTION is applied with no previously checked components, that the Power Source should be checked. The learning process and the learned rules are examined in more detail in the next section.

< Insert Table 1 about here. >

A similar result is obtained as the model consults its interface knowledge with INTERFACE-SUGGESTION [5] and INTERFACE-CHOICE [6]. INTERFACE-CHOICE uses a similar mechanism to the schematic route following, but the representation is based on the order of the components in the interface as shown in Fig. 1a. The power source is suggested to check, and a similar rule (LR-2) is learned to implement the INTERFACE-SUGGESTION operator the next time it is applied and there is not a previous component checked, suggesting that the power source should be checked.

In this example, the suggestions from these two operators examining the diagrammatic knowledge and the interface knowledge concur on the Power Source. These two operators do not necessarily concur—if the last component checked was Energy-Booster 2 (EB2), for example, the next in sequence on the interface is Main Accumulator (MA), whereas the next in the circuit path is a secondary accumulator (SA1 or SA2). This conflict of two different suggested components to check results in an impasse. The SELECT-COMPONENT problem-space and its operators resolve such problems.

The model then starts to check if the light is lit for the Power Source component (TEST-COMPONENT [7] and CHECK-LIT [8] operators in Fig. 3). The process of testing the component entails the gathering of evidence to support or reject the hypothesis that the selected component is faulty. This requires comparisons between the internal representation of the device and its real-world status. The REALITY-CHECK operator [9] leads to noting on the top state the component that needs to be examined.

Deciding to access the state of the world. The next stage is interacting with the (simulated) world and understanding the implications of what was observed. At the start of problem solving,

the model does not know the state of the world and has to look. With the top state annotated with a component to check, the ATTEND operator [10] can be and is proposed. This process of regularly thinking about what component to check and checking gradually constructs a more complete representation of the real world within the model.

Accessing the external world using the top problem space is a way in Soar to avoid learning to do two contradictory things at once (Chong, 2001; Chong & Laird, 1997; John, 1996; Laird & Rosenbloom, 1995; Newell, 1990, p. 261-268). If the model needs to know the status of whether a component is lit or not and its status is unknown, then the REALITY-CHECK operator sets up an ATTEND operator (in the top problem space) to examine the state of the Power Source's indicator light. A rule (LR-3) is learned at this point to propose attending to the Power Source light when the CHECK-LIT operator is selected for the Power Source, but its status is unknown.

Accessing and processing the state of the world. ATTEND [10] and COMPREHEND [11] operators are next used in the top problem space (shown in Fig. 2) to model interaction. This approach is consistent with Newell's (1990, e.g., p. 262-264) suggested use and with Laird and Rosenbloom's (1995) architectural constraints on interaction. Rather than view the entire world at once, the model uses explicit operators to represent acquiring perceptual information. The model continually decides where to look (CHOOSE-COMPONENT), looks there (ATTEND), checks the part it saw (COMPREHEND), and then decides where to look next (if necessary, again with ATTEND). The organization of components on the (memorized) interface schematic and the use of the ATTEND and COMPREHEND operators moving across the representation of the interface cause the components to be checked in a basically left to right sequence. In the case of the EB1 fault (and all problems), Diag starts by attending to the Power Source.

The information on the Power Source is returned by the ATTEND operator [10] and comprehended with the COMPREHEND [11] operator. To do this, TEST-COMPONENT [12] is applied to implement the COMPREHEND operator, and problem solving starts again but this time with the additional information that the Power Source light was on. It is now possible to reapply the CHECK-LIT [13] operator that initiated the attend-comprehend process.

The CHECK-LIT operator [13] this time checks the implications of the light's status (using the DECIDE-STATUS operator [14]). This may require further information about the switches, but for the Power Source, a result can be returned immediately that the Power Source is OK. A rule (LR-4) is also learned that implements the TEST-COMPONENT operator [12] more directly, that if the Power Source light is on and no other components have been tested, then the Power Source is OK.

If the first component was found to be non-faulty, a tidying operator—RESET—is selected before CHOOSE-COMPONENT is selected again. RESET [15] updates the top state to indicate that the component just checked is now the last-checked one.

Repeating this process through the circuit. The process now starts over with the CHOOSE-COMPONENT operator [16], looking for the next component to check, and checking it. There are some different operators used when switches are examined and when multiple paths are found, but the basic process remains the same (a full trace is at the web site). This process continues until a part is found that should be lit and is not lit. Fig. 3 shows this process of ATTEND and COMPREHEND operators (and suboperators) occurring several times for the EB1 fault as the Power Source switch, the Power Source indicator light, the Energy Booster switch, and the Energy Booster 1 light are checked. After checking the Energy Booster 1 light the model confirms the previous component is lit (with the DECIDE-STATUS operator [47]), and then reports the broken component (REPORT [48]).

1.2.3. How and what the model learns

Diag learns while performing the task. What is learned on each trial, when it is learned, and how it is learned arises from the architecture, the knowledge and its organization, and the model's previous problem-solving experience. Learning creates three types of knowledge in Diag. The model learns how to perform actions more directly with less internal search, which objects to attend to in the interface without internal deliberation, and the implications of what it receives from the world. Learning these types of knowledge occur while performing the task. Each of these types of knowledge can be seen individually in other architectures and models.

The three types are: (a) *Operator implementation*, where specific knowledge about how to apply an operator is learned through search in another problem space. Several operators are elaborated in this manner (e.g., INTERFACE-CHOICE, DIAGRAM-CHOICE, and DECIDE-STATUS). This represents a type of procedural learning. Table 1 shows several examples of these learned rules (LR-1, LR-2, LR-4). This learning includes knowledge about how to augment an operator and how to implement it.

(b) *Operator creation*, where an operator is created in one problem space for use in a higher space. The ATTEND operator is created in this way. With practice, the model learns which objects to attend to. It is the transfer and application across faults of the ATTEND operator that give the largest improvements because it takes the most steps to create. These new rules serve as a kind of episodic memory noting the results of problem solving because they contain information

about what problem-solving has already taken place. Table 1 shows two examples of these learned rules (LR-3, LR-9).

(c) *State augmentation* rules, which add to the state derivable knowledge as a type of semantic, declarative learning. For example, the model can learn that if EB1 has been checked (and was working) and the main accumulator's light is not lit, then the MA component is broken. Table 1 shows a similar example for the Energy Booster 1 fault (LR-31).

The number of rules learned in a series of problems varies according to the order, types, and number of tasks. The maximum number of rules that can be learned is 287. On average, the model learns around 200 new rules over the 20 problems in each series of problems, more than doubling the number of rules in the model. When all the rules are learned, the model knows what components to attend to (without deliberation) and the implications of attending to a component (either to attend to the next component or declare the problem solved). The knowledge that is learnt can be applied to later tasks.

The effect of learning is shown in Fig. 4, which shows the order of operator applications the second time the EB1 fault is diagnosed. Fewer operators are used (34 vs. 48), and lower level operators are not used as often (compare with Fig. 3). There is less problem solving between the ATTEND operators because more is known. This is indicated in Fig. 4 through the operator number labels. These labels show the operator numbers from the first trace shown in Fig. 3, and several operators in the sequence are missing because their effects have been learned. For example, the DIAGRAM-CHOICE and INTERFACE-CHOICE operators, 4 and 6, are not necessary in the second run. When no more can be learned, which happens after three trials with a single problem (shown in traces on the web site), behavior consists of a series of ATTEND and COMPREHEND operators until the REPORT-FAULT operator, creating a recognition-based process.

< Insert Fig. 4 about here. >

During the course of solving fault-finding tasks newly learned rules can transfer across trials where the paths checked include the same components. There is no transfer of newly learned rules within a trial because the model does not revisit any components in its problem solving.

2. COMPARISON OF THE MODEL'S PREDICTIONS WITH THE PROBLEM-SOLVING DATA

We show that Diag can account for several regularities that are robust in that they have been found in other studies (Bibby & Payne, 1993, 1996; Bibby & Reichgelt, 1993), as well as being consistent with the data reported here. A number of strategies are used to compare the model against the participants' data. First, we consider whether the structure of the model's behavior is generally the same as that reported by participants in this fault-finding task. Second, linear regression techniques are used to identify which measure of the model's behavior best predicts the participants' problem solving times. Third, the participant's problem solving times are aggregated over task, trial and participants and compared with the model's predictions using the best measure of fit. Finally, individual participant's problem solving times are compared against the model for each individual separately. The first three comparisons are typical of the kinds of model comparison reported in the literature. The final comparison, comparing individual performance against a model that learns automatically with random trial sequences is a novel contribution. This is important because it demonstrates that it is indeed possible to produce fine-grained predictions over a range of learning using an automatic learning mechanism.

2.1 General performance of model and data

The first test of a process model is that it can do the task of interest. The current version of Diag can find any fault in the device given a configuration of lights and switches. More importantly, it implements the same strategy that participants have previously reported. The approximate problem-solving strategy that most participants used is exemplified in a participant's retrospective protocol:

> "...check each component in turn to see if its indicator light is lit. If it is, move on to the next component. If it is not lit, check to see whether it should be lit. If it is not receiving power, move on to the next component. If it is receiving power, then it is broken" (Bibby & Reichgelt, 1993, p. 274).

Diag performs the fault-finding tasks using this strategy. Sequential checking of components from left to right is emergent from the schematic of the internal structure of the device, the interface representation, and the model's knowledge about how to check components and their connections. As we will see in the quantitative comparisons, the model's predictions have a high correspondence with the participants' reaction times.

Participants clicked on the wrong component on 8% of the trials; this is an average of 1.5 errors each. The errors appeared randomly distributed across trials, across tasks, and across participants. The trials where errors were made by participants solving the fault-finding task were removed from all the regressions because the model did not make any errors. We have looked for regularities in these errors, and only found that they are usually close to the correct answer and usually take about the same amount of time to find as the correct answer should have taken. No attempt to model errors was made.

2.2 The choice of prediction variable: Model cycle or interface object

Using hierarchical linear regression, we examine which of three measures of processing time provides the best prediction of problem solving time: (a) the number of interface objects examined, (b) the model's processing time (as decision cycles[6]) with learning turned off, and (c) the model's processing time with learning on. Given that the different examples of the fault-finding task require participants to examine different numbers of components and switches, the first measure of processing time examined was a count of the minimum number of objects that a participant had to examine based on a simple task analysis. This measure is similar to the measure successfully used by Kieras (1992) to predict the time taken to use a device averaged across learning. A second predictor of processing time is derived from the model itself. The Soar environment allows the modeler to run a model with learning switched on or off over the same series of problems that the participants saw. If a model is run with learning switched off then the modeler can assess the impact of learning on task performance when learning is switched on.

The results of the first regression analysis are shown in Table 2. When entered first, the simple analysis of the task predicts 10% of the variability in the participants' time to solve each correctly answered problem (N=185). When the model's processing time with learning off is added it predicts a further 3.7% of the variability. While entering both these measures accounts for a significant proportion of the variability it is not a substantial proportion. Taken together they predict only about 14% of the total variability in participants' problem-solving time.

When model cycles with learning on is entered, it accounts for an additional 57.7% of the variability in the participants' problem-solving times. This result suggests that cycles with learning on is the best predictor of participant performance.

< Insert Table 2 about here. >

To test whether the simple task analysis or the cycles (with learning off) added any extra

explanation beyond that offered by the model through cycles with learning on, the order of entry into the regression equation was changed so that model time with learning on was entered first into the hierarchical regression. This is reported in Table 3. Like the first hierarchical linear regression, a simple task analysis prediction based on the number of objects examined was entered but this time entered second and cycles with learning off was entered last.

When the cycles with learning are used alone to predict the variability in problem-solving time it accounts for 71.5% of the variability and neither the simple task analysis nor the cycles without learning accounts for any additional variability in participants' problem-solving time regardless of their order of entry into the equation. We also examined entering model cycles as the second variable and simple task analysis third. This had nearly identical results and is therefore not reported here.

< Insert Table 3 about here. >

We thus use model cycles with learning as the basis of our predictions. This provides a slope of 128 ms per cycle in the model and an intercept of 3.328 s to account for the time to move the mouse and click on the fault.

2.3 The model's predictions tested

The following analyses examine the correspondence between the model's predictions and the problem-solving time data, and thus where the model could be improved. There are several comparisons available. First, the model may fail in fitting the general learning profile. Second, the model may fail in matching specific examples of the task (e.g., SA2). Third, the model may fail in modeling individual participants. For each of these analyses the predicted times (in seconds) (as 128 ms * model cycles + 3.328 s) are compared with the observed problem-solving times.

Fig. 5 shows the comparison between the predicted and observed average times and standard errors in seconds for each fault aggregated over participants and trials for model and data. For both the predicted and observed times there is a general increase in time taken to solve the problems from left to right across the device interface. Overall, the pattern of similarity between the predicted and the observed problem-solving times by problem type is striking[7]. The model fits the data well with respect to type of fault. As Fig. 5 shows, the overlap between the predicted and observed distributions of time is substantial for each of the types of faults, and the correlation is 0.99.

< Insert Fig. 5 about here. >

Fig. 6 shows that there is a strong relationship between the times the model predicted over the series of 20 trials and the time participants took to solve the problems when trials are aggregated over participants and faults. A learning curve is usually a monotonically decreasing curve (Newell & Rosenbloom, 1981). These data, while highly correlated (r=0.99), are not monotonically decreasing. A post-experiment analysis of the series of faults indicated that there was a problem with the random assignment of faults, particularly that the first two faults were not randomly distributed, appearing on average early in the circuit. To obtain a smooth, monotonically decreasing curve, the average difficulty of faults must remain uniform across the series (Nerb et al., 1999). However, we know that the different faults are not equally difficult. The model and the participants both found that faults immediately after the first two faults in each series took longer to solve because they are later in the circuit. Overall, the model's predictions about learning across trials are well matched.

< Insert Fig. 6 about here. >

The third way in which the model may match the data is its degree of fit for individual participants. Each participant saw a different order of the 20 problems. Fig. 7 shows the total problem-solving time per participant based on the 10 different series of problems that the 10 participants saw, and the predicted times aggregated over these sets of trials and faults. It is apparent that Participant 5's problem-solving times are substantially below those predicted.

< Insert Fig. 7 about here. >

To explore differences between participants further, each set of model cycles per run of the model was regressed against the problem-solving times for each participant individually and the correspondence was graphed. Table 4 shows that the average proportion of variability in problem-solving time per participant accounted for by model cycles was 79%. However, the regression was not significant for two of the participants (P5 and P7). When these participants are removed from the analysis the average proportion of variability increased to 95%. A second check on this result is to examine the B coefficient that represents the number of seconds per model cycle. According to this analysis P5 has a rate of 10 ms per cycle and P7 has a rate of 50 ms per cycle. Both of these values are substantially lower than the average slope (B) coefficient (140 ms/cycle) of the individual regressions or the result of the global regression (128 ms/cycle). P5 is performing quickly and with little learning; P7 is harder to characterize, but may be using a different strategy than what was taught.

We examined how the fit varied across time for the eight participants for whom Diag's

model cycles provided a good fit to their problem-solving times. Fig. 8 shows these fits[8]. The model's predicted times used here are the model cycles adjusted to the average time per model cycle for each participant. These fits are compelling, in that the model and data are close across a range of different series of faults across eight participants. As a group, these plots show substantial amounts of learning, of transfer, and of correspondence between the model's predictions and the participants' performance.

Fig. 9 shows the plots of the predicted and observed problem-solving times for the two participants whose problem solving time was not well predicted by Diag. On these plots the general regression coefficients are used to create the predictions because the individual regression was not significant. For Participant 5 the predicted times are always longer and sometimes substantially so; they appear to know how to perform that task and do not improve. For Participant 7 there is no clear pattern with the participant sometimes taking more time than predicted and sometimes taking less time than predicted.

< Insert Table 4, Fig. 8, and Fig. 9 about here. >

2.4 Summary

In summary, the model appears to implement the same strategy as most participants performing this task. Diag's ability to model the strategy emerges from the structure of the representations of the task environment, both the device interface and the instructions, and the operators that search through the problem spaces. Following the traditional Soar approach of predicting problem-solving times on the basis of model cycles, it was found that Diag's predictions closely fit the aggregated data when examining the specific fault task and the learning profile over 20 trials. The version of the model that best predicted the participants' performance was the Diag model with learning. It provided substantially better predictions than Diag without learning or a simple task analysis. Moreover, Diag was successful at modeling the participants' performance when aggregated over faults and trials. Individual regression analyses indicated that the model fit 8 out of 10 participants' problem-solving times very well. The plots of individual fits show a close correspondence based on learning and transfer.

Diag failed to model two participants. Their percent correct was not significantly worse than average. This suggests that they had adopted an equally successful problem-solving strategy but not the one that Diag uses.

3. DISCUSSION AND CONCLUSIONS

In this paper, we have presented a model that both learns and transfers its learning, exhibiting significant and human-analogous performance time decreases (in some cases by an order of magnitude). The model a does not merely accurately predict problem-solving time for human participants; it also replicates the strategy that human participants use, and it learns at the same rate at which the participants learn. To achieve these three dimensions (time, strategy, and learning rate) of significant similarity to several different human participants shows that the model and architecture represent well-founded theories of human cognition.

We tested the model's predictions using several different measures of participant/model agreement that varied in grain size. The model's predictions match aggregated data fairly well at three different levels: task type, trial number, and total time by participant. The model predictions were also compared against each individual participant's problem-solving times. We found that the match for eight participants was very good (the model's predictions line up with the individual problem solving times with correlations above 0.9); and it modestly matched one participant's ($r^2 = 0.18$) and failed to match one participant's problem-solving times ($r^2 = 0.07$).

The model incorporated an automatic learning mechanism that successfully reproduced individual paths of learning. The model allows us to see where and how expertise arises through repeated problem solving at a fine grain of analysis. The comparison of the model's performance with human behavior shows that the model is a useful representation of what most participants do, learn, and transfer in this task. How the model achieves this provides a number of important insights into the cognitive processes involved in learning and raises issues that could usefully be considered by models in the future. We discuss these issues next.

3.1 Several different types of learning contribute to the learning curve

How does the model learn? When the model is examined in detail it is clear that there are several different kinds of learning taking place, even though this task is relatively simple. Fig. 10 helps describe the learned rules by showing the initial rules and all of the learned rules. Each problem space starts with some initial rules (the white blocks). The operator augmentation and operator result rules are a type of procedural learning. The operator-creation rules can be seen as episodic learning, and the state augmentation rules can be seen as declarative learning.

< Insert Fig. 10 about here. >

Learning in Soar is typically viewed as procedural. Performance improvement through the

acquisition of new rules to implement operators, where specific knowledge about how to apply an operator is learned through search in another problem space, has been used to explain the speed up in performance on a wide variety of tasks (e.g., Seibel-Soar and R1-Soar, Newell, 1990). The fault-finding model implemented here takes advantage of this architectural approach. Such procedural learning accounts for a major part of the learning that takes place in Diag (208 rules, or 72.5% of the learned rules), as shown in Fig. 10.

Another kind of learning produced by the model is a type of episodic learning. As the model attends to different objects in the world it learns which objects it has looked at, which objects it should look at next, and what to do if an object is presently in a specific state. This type of episodic learning is a produced through the ATTEND and COMPREHEND operators in Soar. As rules are built that encode the information about what has been looked at and what should be looked at next, the model is effectively building a trace of its own problem solving. This trace is dependent on the interaction between the internal states of the model and the external world that those internal states represent. If the model did not need to attend to objects in the external world, it would not build the episodic rules that it needs to manage its attentional demands. This learning is in some ways similar to Able (Larkin, 1981) and Able-Soar (Ritter, Jones, & Baxter, 1998). Episodic learning accounts for fewer of the generated rules (70, or 24.4%), but many are in the top problem space so they are useful because they save extensive amounts of work.

Finally, the model also learns to construct declarative knowledge structures that add information to working memory directly summarizing the results of prior problem solving. When that problem state is experienced a second time, a rule fires that simply augments the state with the previous results. There are relatively few of these rules (9, or 3.1%). They appear late in learning, but (as in human learning) such recall provides significant speedup when problem solving is already relatively fast.

The procedural, episodic, and declarative learning that take place all contribute to the speed up in performance on this task and appear necessary to predict transfer of learning between problems in this task. The model predicts that procedural, episodic, and declarative learning all play an important part in the speed up in performance. Given the high degree of fit between the model and the participants' behavior, failing to implement any of these types of learning leads to less accurate predictions.

For example, the first time Diag solves the Laser Bank fault (the most complex task), it takes 332 cycles and learns 57 rules. On the second solution, it solves it in 185 cycles and learns 38 more rules, and on the third it finds the fault in 88 cycles and learns 29 rules. When fully

learned (on the fourth trial), Diag solves the task in 14 cycles.

By artificially teasing apart each of these learning mechanisms, we can see the importance of each mechanism. We performed two such example analyses; first, we ran the model with each of the types of learned rules included separately, and then we ran the model with each of the types removed. If the 208 learned procedural rules (operator augmentation and operator result) alone are included with the initial model, the model finds the Laser Bank fault in 194 cycles. If only the 70 learned episodic rules (operator creation) are included with the initial model, the model cannot solve the problem because the goal stack is left in an inconsistent state, unsurprising, because these operators are predicated on prior experience, and thus do not apply to novel situations. If only the 9 learned declarative rules (state augmentation) are included with the initial model, the model finds the fault in 322 cycles, only 10 cycles faster than the initial run of the model without learning.

The second way is to examine removing each kind of learned rule from a fully learned model. If only the learned procedural rules are removed, the model cannot solve the problem (again, basically due to unusual goal structures); if only the episodic rules are removed, it takes 189 cycles (instead of 14), and if the declarative rules are removed, it takes 19 cycles (instead of 14). Each rule type makes an important contribution, but the savings are of different sizes and come at different points in problem solving, as in human problem-solving. This analysis again makes it clear that to fit individual participant data accurately it is likely that procedural, episodic, and declarative learning all need to be modeled simultaneously, and that learning types interact. At least part of the reason that other models may not attain such good fits to their respective data is that they tend to focus on only one kind of learning. Here, all the rules are needed to solve the problem in 14 cycles.

It is important to remember that in this model, although different kinds of learning take place, a single learning mechanism is responsible for that learning. The differences in the types of learning depend strongly on the requirements of the task itself and the initial representation of the task prior to learning. To complete the task participants need to recall declarative information about the structure on the environment and remember where they are in the task, thus requiring the use of episodic information. The proceduralisation of the ensuing behavior is a natural consequence of the learning mechanism.

The close fit of this theory to the learning data provides a possible explanation of the shape of learning curves and their variance—this model suggests that performance does not speed up according to a simple curve, but according to how much previous learning transfers on each trial.

It also explains why and how the model and participants are sensitive to the order of problem presentation (Ritter, Nerb, O'Shea, & Lehtinen, 2007). While many theories posit this effect, Diag makes a more detailed and substantiated prediction. Previous studies attempt to use the same task throughout, for example, the same set of light patterns across blocks (e.g., Seibel, 1963); other studies have some variation in the task across the series where the learning is measured, for example, different job-shop tasks (Nerb, Ritter, & Krems, 1999), writing different books (Ohlsson, 1992), playing solitaire with shuffled decks (Newell & Rosenbloom, 1981), and solving similar yet different math problems (Neves & Anderson, 1981). If the tasks become harder (or much easier) in the course of a series of trials, Diag predicts that for individual series the power law of learning will not be obtained and instead a more complex curve will be seen. Indeed, our work predicts that even the Seibel task would vary by which pattern of lights was presented each time due to ordering effects (a brief analysis is presented in Nerb, Ritter, & Langley, 2007). The series of problems shown in Fig. 7 illustrate this.

The results in Figure 7 suggest that (at least in some cases) the variation in performance is not noise, but is differential transfer of learning across problems of similar surface features but different difficulty. (Where noise appears within a series of the exact same task, such as cigar rolling (Snoddy, 1926), Diag suggests that perhaps the tasks are not the same for the subject.)

When these problems are averaged together, the power law appears showing the importance of a more detailed analysis. These predictions are consistent with several cognitive theories of transfer, including those by Singley and Anderson (1989), and Kieras and Polson (1985). We have shown that a Soar model can compute the transfer while learning at the same rate as many individual participants, and that a finer grained analysis this model predicts a non power-law learning rate on a trial-by-trial basis.

3.2 Reasoning with diagrams

From the model, we can derive some implications about how participants reason with diagrams and interfaces. The model supports the idea that problem solvers use the memorized diagram information in a cyclical, iterative fashion, even after some practice at the task. Diag also suggests that participants, like the model, learn to use only the information from the diagram that is immediately relevant to the context of each stage of problem-solving. While they use the diagram to solve problems, they learn to apply only that part of the diagram necessary for the task; they do not learn, for example, the reverse path through the circuit.

The model and the participants both use the internal representations and the external user

interface to support problem-solving. Diag and the participant data suggest that after practice, performance of a similar task is driven by recall of each object attended to in a process called recognition-driven problem-solving (Chase & Simon, 1973; Klein, 1989).

One implication of Diag is that if a new task were presented to a trained model, it will need more problem-solving on a new task. This fits with findings reported by Bibby and Payne (1993, 1996), who looked at a range of problem-solving task (including fault-finding) and instructional materials (including diagnostic tables and diagrams). They found that over a period of 80 trials, differences in performance between participants who learned from different instructional materials disappeared, only to re-emerge when participants were presented with new tasks.

Diag can explain these results. As expertise in the task increases the reliance on the instructions (in the current case the diagram) decreases. It is important to note that the problem space that represents the diagram is not lost to the model; merely the need to refer to it directly decreases with practice. When a new task is presented, the model is forced to refer back to the diagram to extract the relevant information. The current modeling enterprise suggests an explanation of how the long term consequences of receiving particular instructions emerge from the requirement to re-use the instructional representations in new contexts to solve new tasks, and it provides an implementation of Rasmussen's (1983) levels of knowledge.

In complicated tasks, particularly for tasks that use both internal and external representations of the task, forcing the model to interact to obtain information makes information-seeking behavior to be treated as an explicit task. Including interaction as a task helps bring the model timing into agreement with participants (Byrne & Kirlik, 2005; G. Jones & Ritter, 2000; St. Amant, Horton, & Ritter, 2007; Taatgen & Lee, 2003). This approach to interaction is consistent with ACT-R/PM (Byrne, 2001), EPIC (Kieras & Hornof, 2004), and EPIC-Soar (Chong & Laird, 1997) in that these approaches have models interact with tasks through simulated perception and motor action.

3.3 Architecture timing predictions

Having a good measure of the processing rate of the model is important for making *a priori* predictions of reaction times and the time course of problem-solving behavior based on the model. Such predictions can assist with the complicated task of fixing the architecture's timing predictions due to many variables, including differences in participants, tasks, task strategies, process models, and the architecture itself.

Soar was originally designed to have a model cycle taking approximately 100 ms. Newell intended that to mean within an order of magnitude around 100 ms, more specifically, between 30 and 300 ms (Newell, 1990, p. 224, expresses a cycle as ~~100 ms, and on p. 121 this double tilde is defined as "times or divided by 3"). Thus, the average predicted cycle time of 140 ms observed in this comparison fits well within these bounds. Soar models of simple tasks often come quite close to matching data with a 50-100 ms cycle without adjustments. These include simple reaction times (70 ms, Newell, 1990, p. 273 et seq., but with a modified, minimum operator time), covert visual attention (50 ms, Wiesmeyer, 1991, p. 48), and performing the Wicken's dual-task (50 ms: Chong, 1998, Appendix F).

Where Soar models have been compared with more complex behavior, the model cycle rate is usually greater than 100 ms (313 and 368 ms, Nerb, Ritter, & Krems, 1999; 145 ms, Peck & John, 1992 analyzed in Ritter, 1992, p. 146). Situations where the cycle corresponds to a longer period of time can indicate that the model is too intelligent, in that it is performing the task more efficiently than participants (and thus taking less cycles because of a more efficient strategy) or not performing as much of the task as the participants (e.g., not modeling interaction, errors, or search). This situation is better than having the cycle time being too short, where the model is performing more information processing than the participant (Kieras, Wood, & Meyer, 1997). The model cycle time reported here may also be unusual because it is one of the first reported for individuals (as opposed to aggregate data). In general, we believe that it will take numerous tasks, strategies, and repeated attempts to fix a proper value for parameters such as processing rate. The work reported here fits very well within the range and helps to further define this time.

3.4 Conclusions

Constructing a model that reproduces individual data during learning helps us examine how cognitive skills develop. We have shown that it is possible to construct a model using a general learning mechanism that can capture learning at the rate it occurs on an individual, trial-by-trial level. This approach contrasts with previous models of learning during problem solving, which have often succeeded by hand-crafting rules and inserting them into the model as the problem solving progresses. Hand-crafting rules gave the cognitive modeler a high degree of control, but at the cost of explaining how such rules arise.

In this paper, a single learning mechanism embedded within a cognitive architecture, Soar, has proven itself sufficient for performing the modeled task, matching aggregate data, and matching and sequential data at a detailed level, including the rate of individuals learning over a series of 20 trials. Although there is a single learning mechanism, it is important to note that this

single mechanism produced several *types* of learning: through practice, the model acquires procedural, episodic, and declarative knowledge . The close fit with the data suggests that the model's ability to acquire, transfer, and use this knowledge accounts for how the human participants learned, and shows how learning and problem solving can lead to more recognition-based performance.

It is possible that models in other architectures will be able to duplicate these results. Models will require a variety of knowledge, including where to look, how to choose what to examine (the internal or external representation of the circuit), and how to learn through interaction. They must also include multiple types of learning, which appears to lead to the learning curve in general and to support a theory of transfer based on the individual tasks performed. A model built in a different architecture, but of the same task and matching the same data, would help modelers better understand and differentiate among architectures and their learning mechanisms. We welcome such contributions—as mentioned earlier, our data, along with our model, is available on the web.

Acknowledgments

We thank John Anderson, Ellen Bass, Gordon Baxter, Richard Carlson, Mark Cohen, Peter Delaney, Maik Friedrich, Fernand Gobet, Wayne Gray, Joshua Gross, Bonnie John, Sue Kase, Clayton Lewis, Geoff Morgan, Michael Schoelles, Robert Stark, Mark Steadman, Richard Young (several times), participants at the ONR/Darpa/NSF Workshop on Reasoning and Learning at Stanford in 2004, comments at ICCM 2001, and the anonymous reviewers for comments and discussions. Sam Marshall provided extensive comments and programmed the model; Shara Lochun ran the study and assisted with the analysis. Support for this work was provided by a grant from the Joint Council Initiative in HCI and Cognitive Science, number SPG 9018736.

REFERENCES

Altmann, E. M., & John, B. E. (1999). Episodic indexing: A model of memory for attention events. *Cognitive Science, 23*(2), 117-156.

Altmann, E. M., Larkin, J. H., & John, B. E. (1995). Display navigation by an expert programmer: A preliminary model of memory. In *Proceedings of the CHI'95 Conference on Human Factors in Computing Systems,* 3-10. New York, NY: ACM.

Altmann, E. M., & Trafton, J. G. (2002). Memory for goals: An activation-based model. *Cognitive Science, 26*(1), 39-83.

Anderson, J. R. (2007). *How can the human mind exist in the physical universe?* New York, NY: Oxford University Press.

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

Anderson, J. R., Conrad, F. G., & Corbett, A. T. (1989). Skill acquisition and the LISP tutor. *Cognitive Science, 13*(4), 467-505.

Anzai, Y., & Simon, H. A. (1979). The theory of learning by doing. *Psychological Review, 86*, 124-140.

Bauer, M. I., & John, B. E. (1995). Modeling time-constrained learning in a highly interactive task. In *Proceedings of the CHI'95 Conference on Human Factors in Computer Systems,* 19-26. New York, NY: ACM.

Bibby, P. A., & Payne, S. J. (1993). Internalisation and the use specificity of device knowledge. *Human-Computer Interaction, 8*, 25-56.

Bibby, P. A., & Payne, S. J. (1996). Instruction and practice in learning to use a device. *Cognitive Science, 20*(4), 539-578.

Bibby, P. A., & Reichgelt, H. (1993). Modelling multiple uses of the same representation in Soar. In *Prospects for Artificial Intelligence,* 271-280. Amsterdam: IOS Press.

Byrne, M. D. (2001). ACT-R/PM and menu selection: Applying a cognitive architecture to HCI. *International Journal of Human-Computer Studies, 55*(1), 41-84.

Byrne, M. D., & Kirlik, A. (2005). Using computational cognitive modeling to diagnose possible sources of aviation error. *International Journal of Aviation Psychology, 15*(2), 135-155.

Chase, W. G., & Simon, H. A. (1973). Perception in chess. *Cognitive Psychology, 4*, 55-81.

Chong, R. S. (1998). *Modeling dual-task performance improvement: Casting executive process knowledge acquisition as strategy refinement*. Unpublished PhD thesis. CSE-TR-378-98, The University of Michigan.

Chong, R. S. (2001). Low-level behavioral modeling and the HLA: An EPIC-Soar model of an enroute air-traffic control task. In *Proceedings of the 10th Computer Generated Forces and Behavioral Representation Conference,* 27-35. Orlando, FL: Division of Continuing Education, University of Central Florida. www.sisostds.org/cgf-br/10th/.

Chong, R. S., & Laird, J. E. (1997). Identifying dual-task executive process knowledge using EPIC-Soar. In *Proceedings of the 19th Annual Conference of the Cognitive Science Society,* 107-112. Mahwah, NJ: Erlbaum.

Feigenbaum, E. A., & Simon, H. A. (1984). EPAM-like models of recognition and learning. *Cognitive Science, 8*, 305-336.

Gobet, F., & Lane, P. C. R. (2007). An ordered chaos: How do order effects arise in a cognitive model? In F. E. Ritter, J. Nerb, T. M. O'Shea & E. Lehtinen (Eds.), *In order to learn: How the sequence of topics influences learning* (pp. 107-118). New York, NY: Oxford.

Gonzalez, C., Lerch, J. F., & Lebiere, C. (2003). Instance-based learning in dynamic decision making. *Cognitive Science, 27*, 591-635.

John, B. E. (1996). Task matters. In D. M. Steier & T. M. Mitchell (Eds.), *Mind matters: A tribute to Allen Newell* (pp. 313-324). Mahwah, NJ: Erlbaum.

John, B. E., Vera, A. H., & Newell, A. (1994). Towards real-time GOMS: A model of expert behavior in a highly interactive task. *Behavior and Information Technology, 13*, 255-267.

Jones, G., & Ritter, F. E. (2000). Over-estimating cognition time: The benefits of using a task simulation. In *Simulating Human Agents, American Association for Artificial Intelligence Fall 2000 Symposium Series,* 67-74. Menlo Park, CA: AAAI Press.

Jones, G., Ritter, F. E., & Wood, D. J. (2000). Using a cognitive architecture to examine what develops. *Psychological Science, 11*(2), 93-100.

Jones, R. M., & VanLehn, K. (1992). A fine-grained model of skill acquisition: Fitting Cascade to individual subjects. In *Proceedings of the Fourteenth Annual Conference of the Cognitive Science Society,* 873-878. Hillsdale, NJ: Erlbaum.

Kieras, D. E. (1988). Towards a practical GOMS model methodology for user interface design. In M. Helander (Ed.), *Handbook of Human-Computer Interaction*. North-Holland: Elsevier Science.

Kieras, D. E. (1992). Diagrammatic displays for engineering systems: Effects of human performance in interacting with malfunctioning systems. *International Journal on Man-Machine Studies, 36*, 861-895.

Kieras, D. E., & Bovair, S. (1984). The role of a mental model in learning how to operator a device. *Cognitive Science, 8*, 255-273.

Kieras, D. E., & Hornof, A. J. (2004). Building cognitive models with the EPIC architecture for human cognition and performance. In *Proceedings of ICCM - 2004 - Sixth International Conference on Cognitive Modeling,* Mahwah, NJ: Erlbaum. simon.lrdc.pitt.edu/~iccm/proceedings/schedule.htm.

Kieras, D. E., & Polson, P. G. (1985). An approach to the formal analysis of user complexity. *International Journal of Man-Machine Studies, 22*, 365-394.

Kieras, D. E., Wood, S. D., & Meyer, D. E. (1997). Predictive engineering models based on the EPIC architecture for a multimodal high-performance human-computer interaction task. *Transactions on Computer-Human Interaction, 4*(3), 230-275.

Klein, G. A. (1989). Recognition-primed decisions. In W. B. Rouse (Ed.), *Advances in Man-Machine Systems Research, Vol. 5* (Vol. 5, pp. 47-92). Greenwich, CT: JAI.

Laird, J. E. (2003). The Soar Tutorial (Version www.eecs.umich.edu/~soar/getting-started.html).

Laird, J. E., Newell, A., & Rosenbloom, P. S. (1987). Soar: An architecture for general intelligence. *Artificial Intelligence, 33*(1), 1-64.

Laird, J. E., & Rosenbloom, P. S. (1995). The evolution of the Soar cognitive architecture. In D. M. Steier & T. M. Mitchell (Eds.), *Mind matters* (pp. 1-50). Hillsdale, NJ: Erlbaum.

Larkin, J. H. (1981). Enriching formal knowledge: A model for learning to solve textbook physics problems. In J. R. Anderson (Ed.), *Cognitive skills and their acquisition* (pp. 311-334). Hillsdale, NJ: Erlbaum.

Lee, F. J., & Anderson, J. R. (2001). Does learning a complex task have to be complex?: A study in learning decomposition. *Cognitive Psychology, 42*, 267–316.

Lehman, J. F., Laird, J. E., & Rosenbloom, P. S. (1996). A gentle introduction to Soar, an architecture for human cognition. In S. Sternberg & D. Scarborough (Eds.), *Invitation to cognitive science* (Vol. 4). Cambridge, MA: MIT Press.

Lewis, R. L. (1993). *An architecturally-based theory of human sentence comprehension*. Unpublished Ph. D. thesis. CMU-CS-93-226, Carnegie-Mellon University, Pittsburgh, PA.

Lewis, R. L. (2001). Cognitive theory, Soar. In *International encylopedia of the social and behavioral sciences*. Amsterdam: Pergamon (Elsevier Science).

Nerb, J., Ritter, F. E., & Krems, J. F. (1999). Knowledge level learning and the power law: A Soar model of skill acquisition in scheduling. *Kognitionswissenschaft [Journal of the German Cognitive Science Society] Special issue on cognitive modelling and cognitive architectures, D. Wallach & H. A. Simon (eds.), 8*(1), 20-29.

Nerb, J., Ritter, F. E., & Langley, P. (2007). Rules of order: Process models of human learning. In F. E. Ritter, J. Nerb, T. O'Shea & E. Lehtinen (Eds.), *In order to learn: How the sequences of topics affect learning* (pp. 57-69). New York, NY: Oxford University Press.

Neves, D. M., & Anderson, J. R. (1981). Knowledge compilation: Mechanisms for the automatization of cognitive skills. In J. R. Anderson (Ed.), *Cognitive skills and their acquisition*. Hillsdale, NJ: Erlbaum.

Newell, A. (1980). Reasoning, problem solving and decision processes: The problem space as a fundamental category. In R. Nickerson (Ed.), *Attention and performance VIII*. Hillsdale, NJ: Erlbaum.

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. 1-51). Hillsdale, NJ: Erlbaum.

Newell, A., Yost, G. R., Laird, J. E., Rosenbloom, P. S., & Altmann, E. (1991). Formulating the problem space computational model. In R. F. Rashid (Ed.), *Carnegie Mellon Computer Science: A 25-Year commemorative* (pp. 255-293). Reading, MA: ACM-Press (Addison-Wesley).

O'Reilly, R. C., & Munakata, Y. (2000). *Computational explorations in cognitive neuroscience: Understanding the mind by simulating the brain*. Cambridge, MA: MIT Press.

Ohlsson, S. (1992). The learning curve for writing books: Evidence from Professor Asimov. *Psychological Science, 3*(6), 380-382.

Ohlsson, S. (2007). Constraining order: Order effects in constraint-based skill acquisition. In F. E. Ritter, J. Nerb, E. Lehtinen & T. O'Shea (Eds.), *In order to learn: How the sequences of topics affect learning* (pp. 151-165). New York, NY: Oxford University Press.

Pavlik, P. I. (2007). Timing is in order: Modeling order effects in the learning of information. In F. E. Ritter, J. Nerb, E. Lehtinen & T. O'Shea (Eds.), *In order to learn: How the sequences of topics affect learning* (pp. 137-150). New York, NY: Oxford University Press.

Peck, V. A., & John, B. E. (1992). Browser-Soar: A computational model of a highly interactive task. In *Proceedings of the CHI '92 Conference on Human Factors in Computing Systems,* 165-172. New York, NY: ACM.

Ram, A., Narayanan, S., & Cox, M. T. (1995). Learning to troubleshoot: Multistrategy learning of diagnostic knowledge for a real-world problem-solving task. *Cognitive Science, 19*(3), 289-340.

Rasmussen, J. (1983). Skills, rules, knowledge: Signals, signs and symbols and other distinctions in human performance models. *IEEE Transactions: Systems, Man, & Cybernetics, SMC-13*, 257-267.

Ritter, F. E. (1992). *TBPA: A methodology and software environment for testing process models' sequential predictions with protocols*. Unpublished PhD thesis, Carnegie-Mellon University, Pittsburgh, PA.

Ritter, F. E. (2003). Soar. In L. Nadel (Ed.), *Encyclopedia of cognitive science* (Vol. 4, pp. 60-65). London: Nature Publishing Group.

Ritter, F. E., Jones, R. M., & Baxter, G. D. (1998). Reusable models and graphical interfaces: Realising the potential of a unified theory of cognition. In U. Schmid, J. Krems & F. Wysotzki (Eds.), *Mind modeling—A cognitive science approach to reasoning, learning and discovery* (pp. 83-109). Lengerich, Germany: Pabst Scientific Publishing.

Ritter, F. E., & Larkin, J. H. (1994). Developing process models as summaries of HCI action sequences. *Human-Computer Interaction, 9*(3), 345-383.

Ritter, F. E., Nerb, J., O'Shea, T., & Lehtinen, E. (Eds.). (2007). *In order to learn: How the sequences of topics affect learning*. New York, NY: Oxford University Press.

Ritter, F. E., & Young, R. M. (1996). The Psychological Soar Tutorial, acs.ist.psu.edu/nottingham/pst-ftp.html (Version Vers. 12) [HTML document]. Nottingham: Psychology Department, U. of Nottingham.

Rosenbloom, P. S., & Newell, A. (1987). Learning by chunking, a production system model of practice. In D. Klahr, P. Langley & R. Neches (Eds.), *Production system models of learning and development* (pp. 221-286). Cambridge, MA: MIT Press.

Ruiz, D., & Newell, A. (1989). Tower-noticing triggers strategy-change in the Tower of Hanoi: A Soar model. In *Proceedings of the Eleventh Annual Conference of the Cognitive Science Society,* 522-529. Hillsdale, NJ: Erlbaum.

Seibel, R. (1963). Discrimination reaction time for a 1,023-alternative task. *Journal of Experimental Psychology, 66*(3), 215-226.

Singley, M. K., & Anderson, J. R. (1989). *The transfer of cognitive skill*. Cambridge, MA: Harvard University Press.

Snoddy, G. S. (1926). Learning and stability. *Journal of Applied Psychology, 10*, 1-36.

St. Amant, R., Horton, T. E., & Ritter, F. E. (2007). Model-based evaluation of expert cell phone menu interaction. *ACM Transactions on Computer-Human Interaction, 14*(1), 24 pages.

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

Taatgen, N. A., & Lee, F. J. (2003). Production compilation: A simple mechanism to model complex skill acquisition. *Human Factors, 45*(1), 61-76.

Thimbleby, H. (9 May, 2004). "Give your computer's IQ a boost," Review of Journal of Machine Learning Research. *Times Higher Education Supplement*.

VanLehn, K. (1989). Learning events in the acquisition of three skills. In *Proceedings of the 11th Annual Conference of the Cognitive Science Society,* 434-441. Hillsdale, NJ: Erlbaum.

VanLehn, K. (1991). Rule acquisition events in the discovery of problem-solving strategies. *Cognitive Science, 15*(1), 1-47.

VanLehn, K., & Jones, R. M. (1993). Learning by explaining examples to oneself: A computational model. In S. Chipman & A. L. Meyrowitz (Eds.), *Foundations of knowledge acquisition: Cognitive models of complex learning* (pp. 25-82). Boston, MA: Kluwer.

Wiesmeyer, M. D. (1991). An operator-based attentional model of rapid visual counting. In *Proceedings of the Thirteenth Annual Conference Cognitive Science Society,* 552-557. Hillsdale, NJ: Erlbaum.

Table 1. Examples of the rules learned when solving the EB1 problem repeatedly, summarized from the model trace in the online file eb1-trace-bottomup.txt.

LR-1: IF the DIAG-SUGGESTION operator is applied to a state that has not had any components checked, THEN suggest checking the Power Source. (Procedural, operator implementation)

LR-2: IF the INTERFACE-SUGGESTION operator is applied to a state that has not had any components checked, THEN suggest checking the Power Source. (Procedural, operator implementation)

LR-3: IF the CHECK-LIT operator in the Test-component problem space is to be applied to the Power Source and the Power Source light's value is not known, THEN propose an ATTEND operator in the top problem space that will check the Power Source's light. (Episodic, operator creation)

LR-4: IF the TEST-COMPONENT operator is to be applied to the Power Source and the Power Source light's status is that it is lit, THEN set the component-status to be OK. (Procedural, operator implementation)

LR-9: IF the CHECK-LIT operator in the Test-component problem space is to be applied to the Energy Booster 1 and the Energy Booster light's value is not known, THEN propose an ATTEND operator in the top problem space that will check the Energy Booster 1's light. (Episodic, operator creation)

LR-31: IF the Energy Booster 1 is the component being checked, and it has been checked, and it is not lit and it should be lit, and the previous component checked was the Power Source, THEN the Energy Booster 1 is faulty. (Declarative, state augmentation)

Table 2. The results of a hierarchical linear regression for problem-solving time showing the proportion of variability accounted for by each predictor in (forced) order of entry, the change in variability from the previous step, and the statistical significance of that change.

	Intercept	B	Beta	ΔR^2	R^2
Simple Task Analysis	5.169	0.956	0.317	.100**	.100**
Model Cycles (Learning Off)	1.402	0.121	1.082	.037*	.138**
Model Cycles (Learning On)	2.645	0.127	0.840	.577**	.715**

*p < 0.05, **p < 0.001

Table 3. The results of a hierarchical linear regression for problem-solving time showing the proportion of variability accounted for by each predictor in (forced) order of entry, the change in variability from the previous step, and the statistical significance of that change.

	Intercept	B	Beta	ΔR^2	R^2
Model Cycles (Learning On)	3.238	0.128	.845	.714**	.714**
Simple Task Analysis	3.261	0.000	-.001	.000	.714**
Model Cycles (Learning Off)	2.645	-0.671	-.222	.001	.715**

**p<0.001

Table 4. The degree of fit (R^2), intercept (in seconds), and regression coefficient (B, in seconds per model cycle) when cycles is regressed against problem-solving time, and the number of correct responses for each individual participant.

Participant	R^2	Intercept	B	N
P1	0.92**	1.87	0.14	19
P2	0.98**	2.12	0.17	16
P3	0.94**	2.12	0.18	19
P4	0.97**	2.09	0.15	20
P5	0.07	4.10	0.01	20
P6	0.95**	2.27	0.18	18
P7	0.18	9.50	0.05	19
P8	0.94**	4.19	0.12	19
P9	0.98**	2.53	0.15	17
P10	0.97**	1.77	0.15	18
Mean	0.79	3.26	0.14	18.5

**$p<0.01$

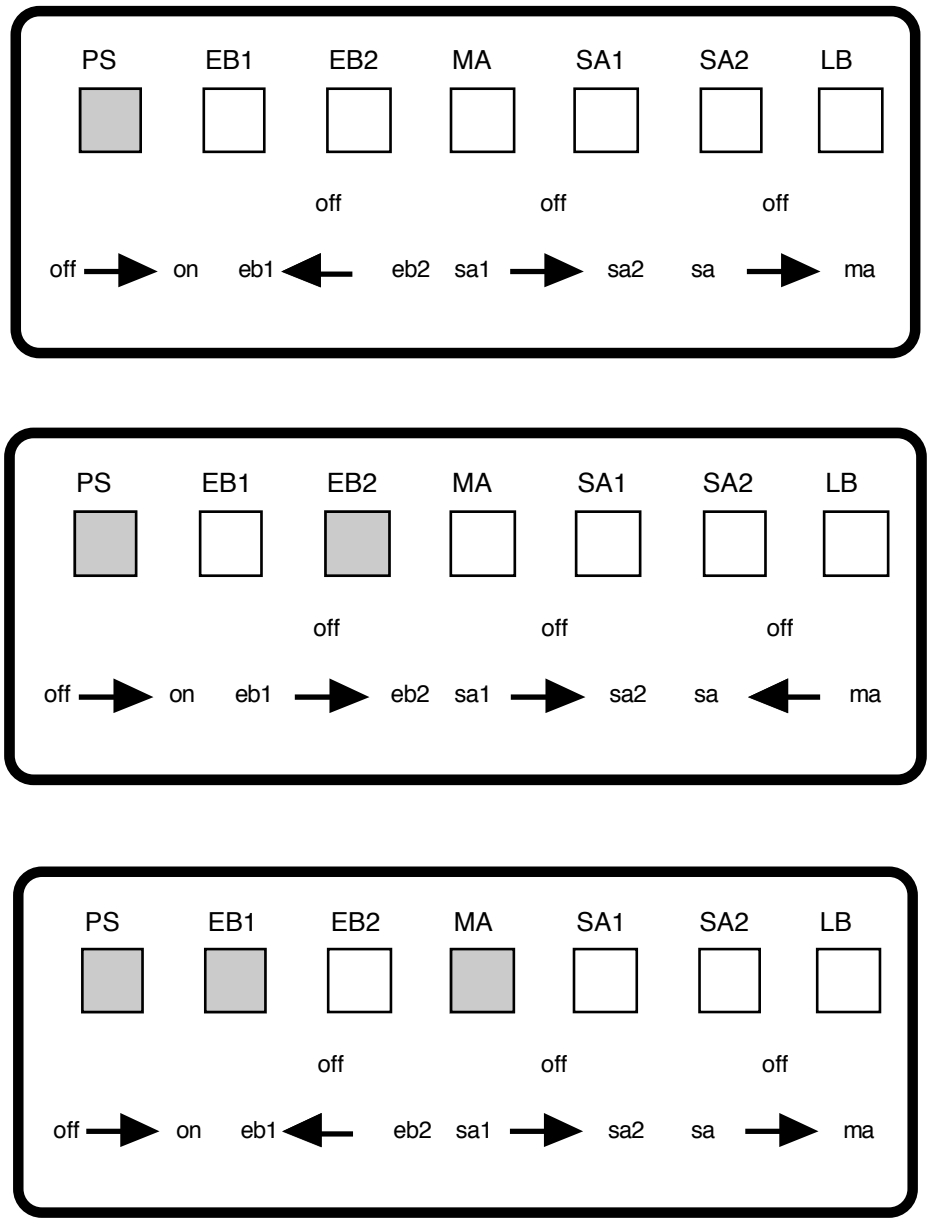


Fig. 1a. The laser-bank interface that the participant used to find the faulty component. Boxes (the lights) below the abbreviations and above the switches (the arrows) when grayed indicate components that are working. The top interface illustrates a component fault (component EB1 is faulty). The middle and bottom interfaces have a Secondary Accumulator Two (SA2) and a Laser Bank (LB) fault, respectively.

Imagine that you are on board a Klingon warship under attack from the Starship Enterprise for attempting to smuggle arms to the planet Orion III, your new allies. Your job is to operate the new laser weaponry developed using designs based on the phasers on board the Enterprise. The laser system has been designed so that it can be made to work when some of the components are broken.

The laser system comprises a power source (PS), two energy boosters (EB1 and EB2), accumulators (MA, SA1 and SA2) and the laser bank (LB).

Power is routed through the system by changing the position of switches directing the power from the power source on to one of the energy boosters then to one of the accumulators and finally an accumulator is selected to send power to the Laser Bank.

If a component is in working order then its indicator light will come on when it is receiving power.

Fig. 1b. General introduction to the device.

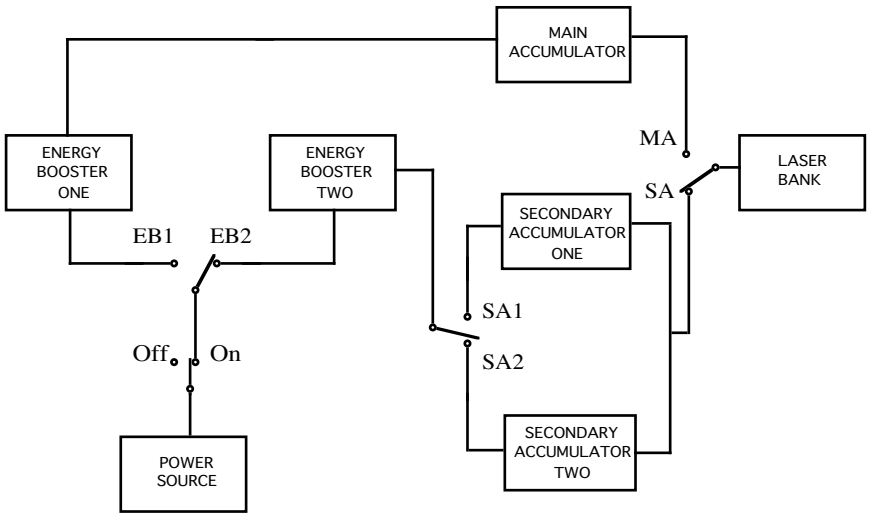


Fig. 1c. Schematic of the laser-bank device. On the interface display, and later in the text and figures of this paper, straightforward abbreviations are used to refer to components in the schematic. For example, PS stands for Power Source, LB for Laser Bank, and the rest are given above on the switches.

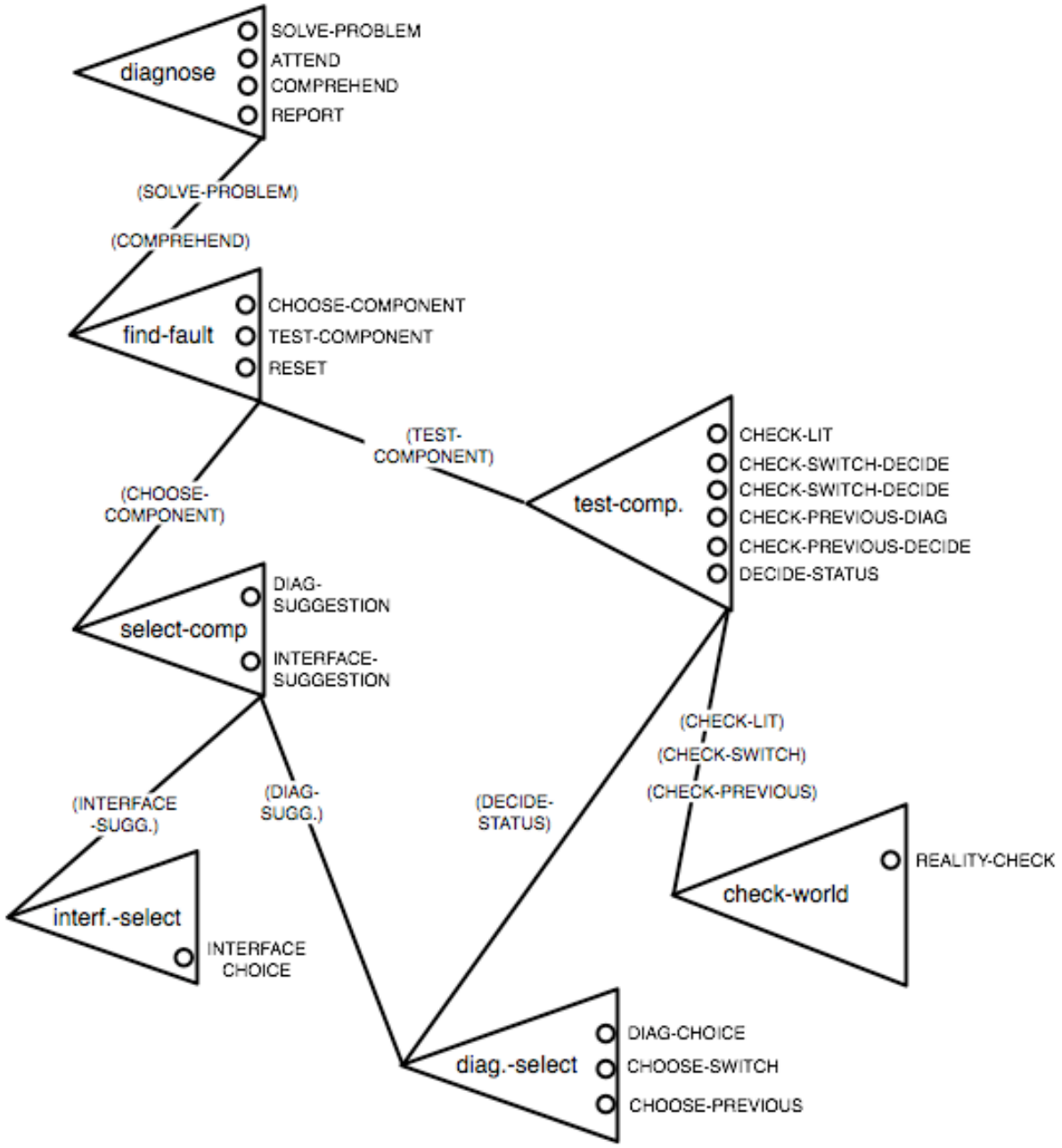


Fig. 2. Organization of the model showing its hierarchical structure. Triangles represent problem spaces; circles represent operators (in small caps). Links to sub-problem spaces to implement operators are represented by a link labeled with the operator name in parentheses.

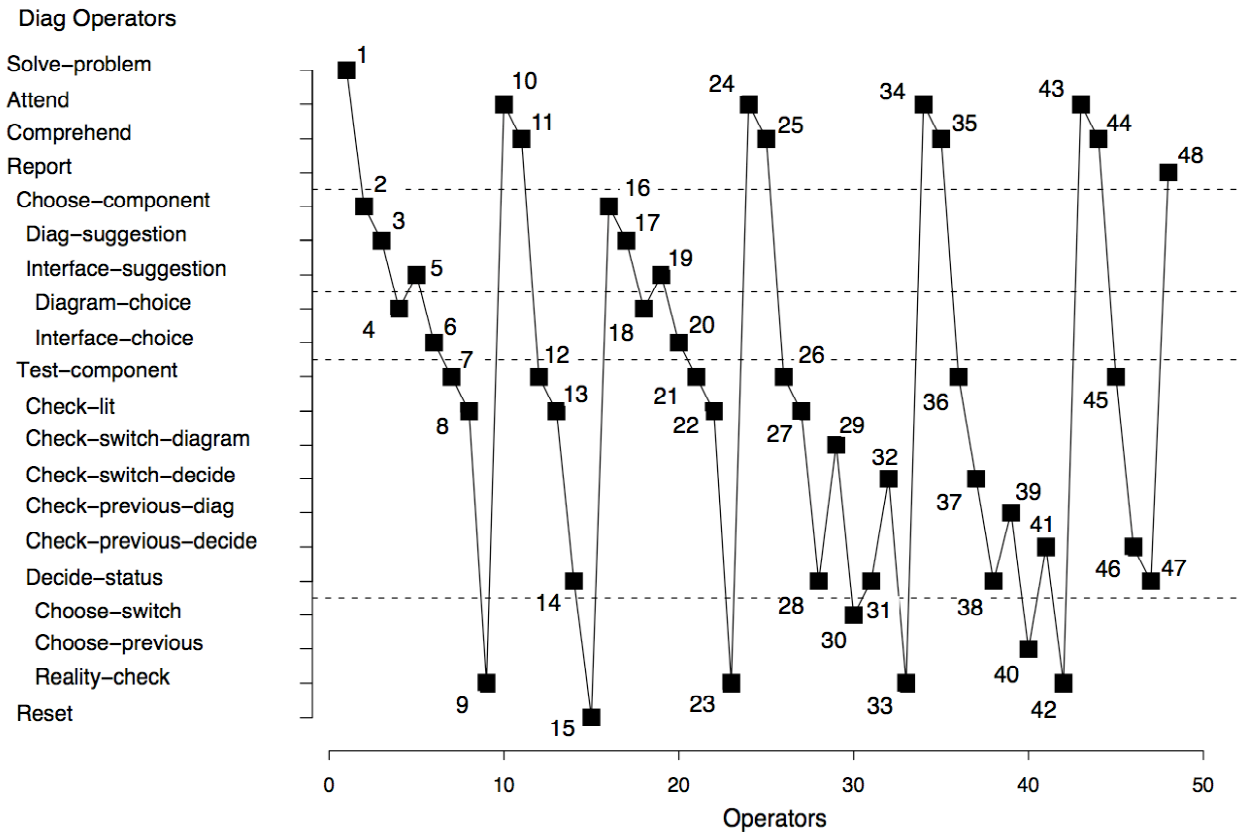


Fig. 3. Operator applications in the model the first time it solves the Energy Booster 1 fault.

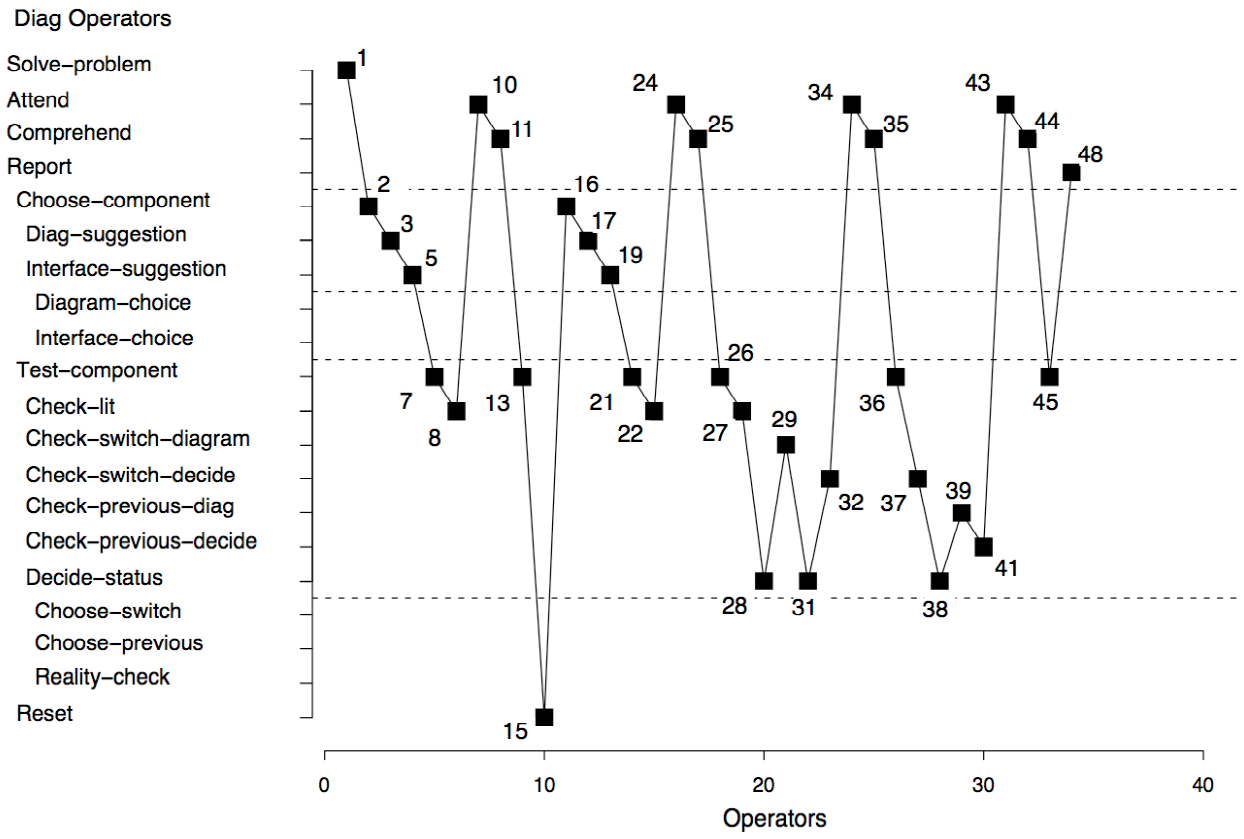


Fig. 4. Operator applications in the model the second time it solves the Energy Booster 1 fault. The numbers on the operators correspond to their use in the previous run shown in Fig. 3.

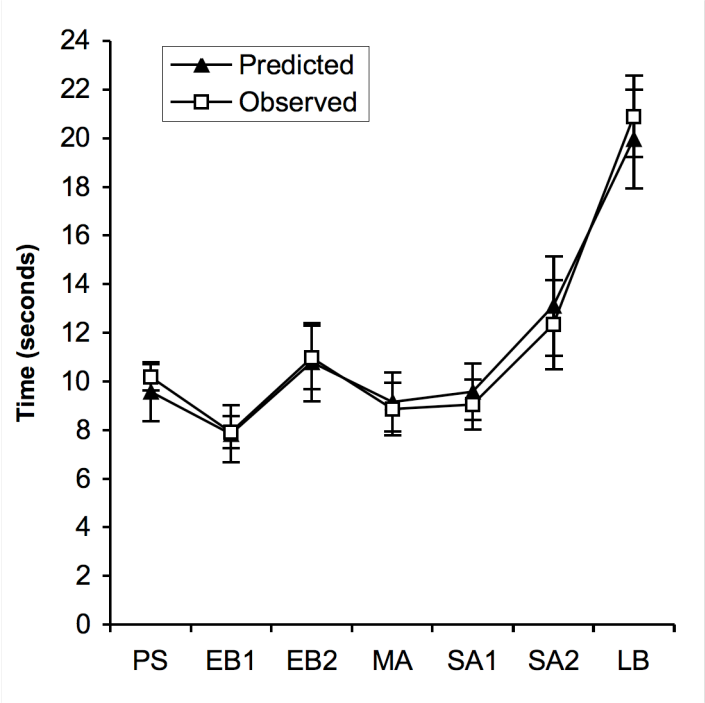


Fig. 5. Predicted and observed problem-solving times (means and standard errors) for the seven different faults averaged over participants and trials.

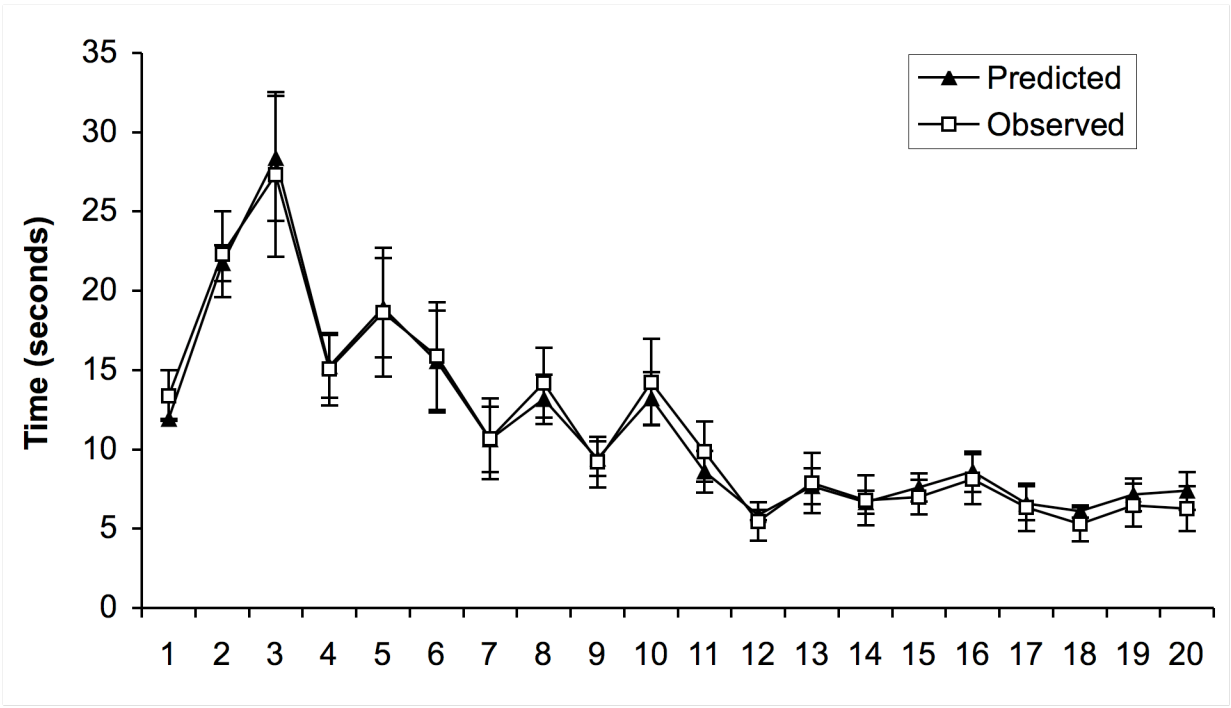


Fig. 6. Predicted and observed problem-solving times (means and standard errors) for the 20 trials averaged over participants and faults.

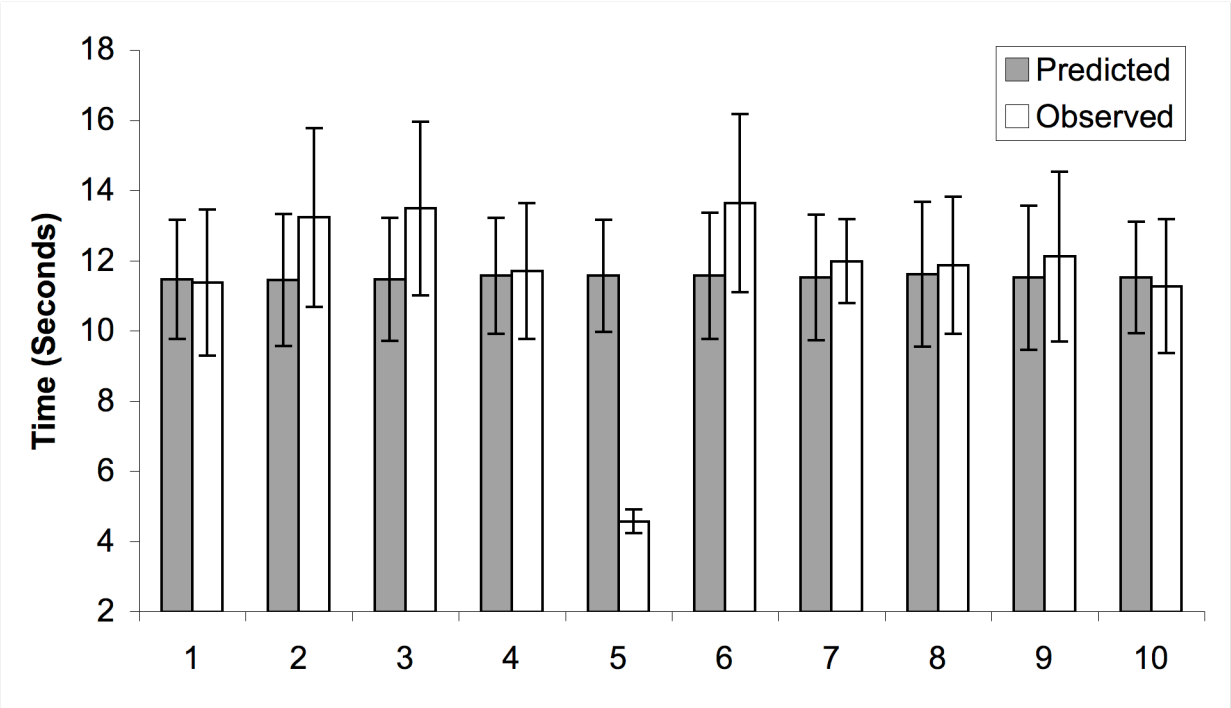
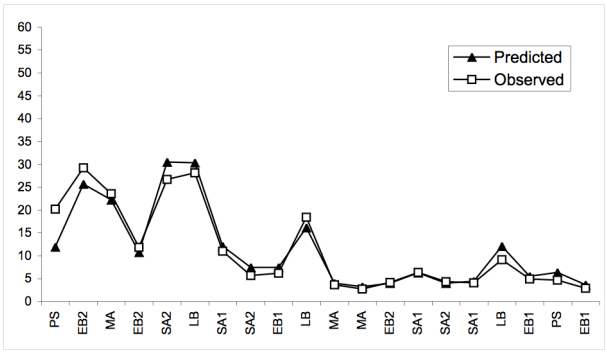
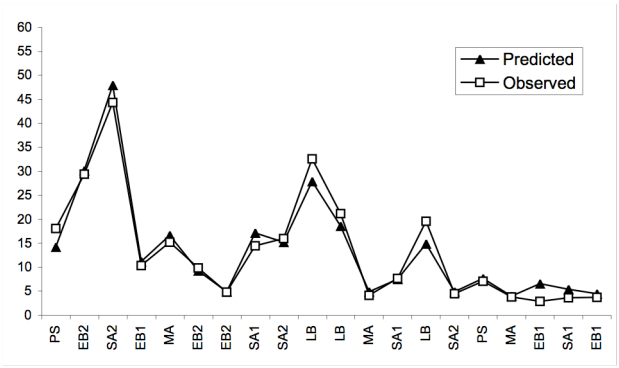


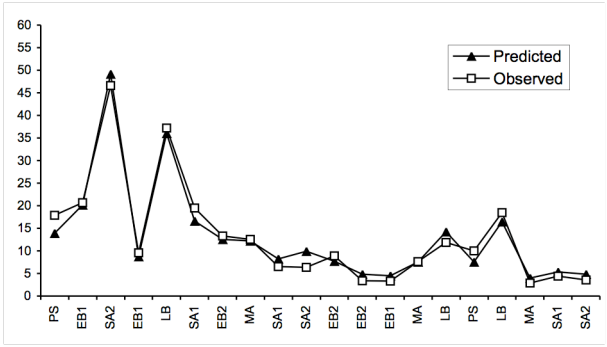
Fig. 7. Predicted and observed average problem-solving times (means and standard errors) for the 10 participants and for the model run on the series each participant saw. (Each participant saw a unique series of problems.)



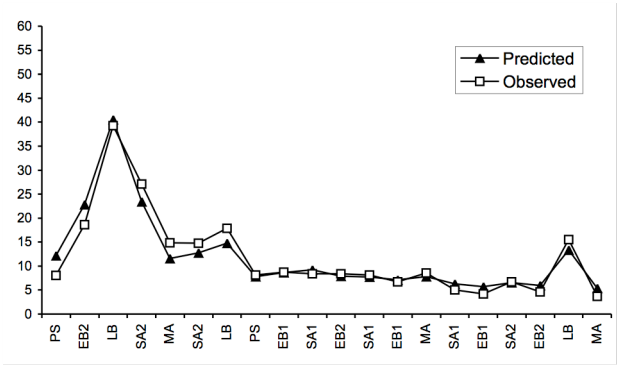
Participant 1



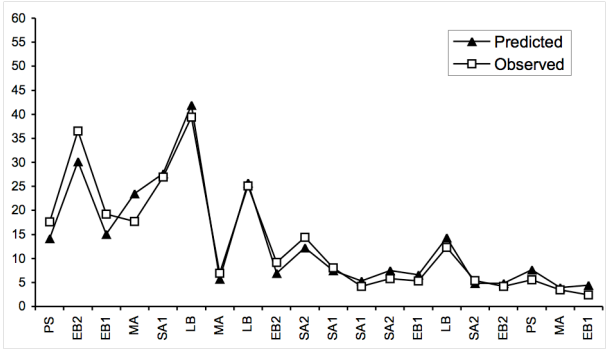
Participant 6



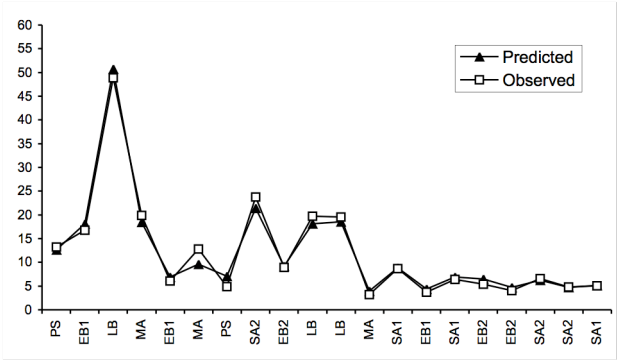
Participant 2



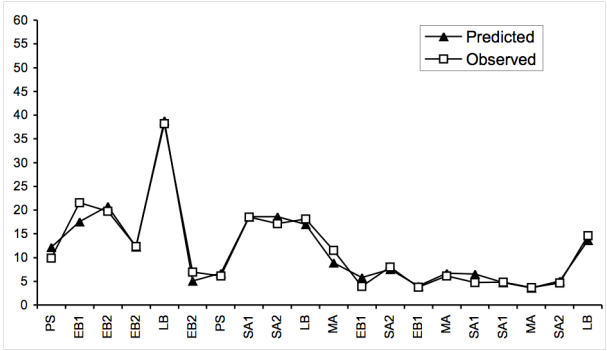
Participant 8



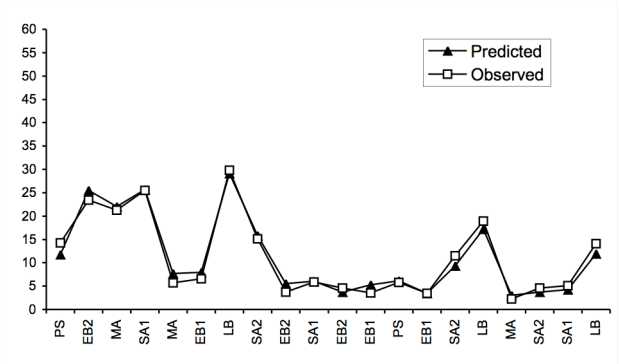
Participant 3



Participant 9



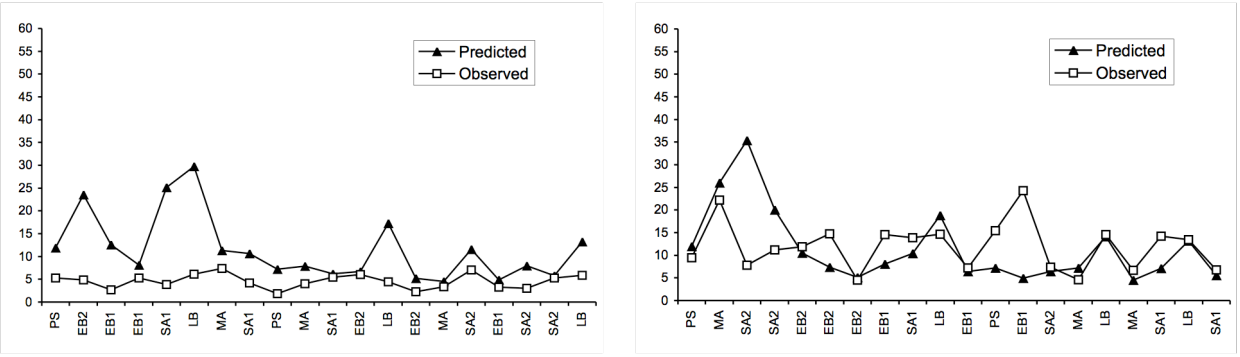
Participant 4



Participant 10

Fig. 8. Predicted problem-solving times (based on individual regressions) and the

observed times in task presentation order for the 8 out of 10 participants for whom Diag's model cycles significantly predict problem-solving time.



Participant 5 Participant 7

Fig. 9. Predicted (using global regression coefficients) and observed problem-solving times in task presentation order for the two participants for whom Diag does not significantly predict problem-solving time.

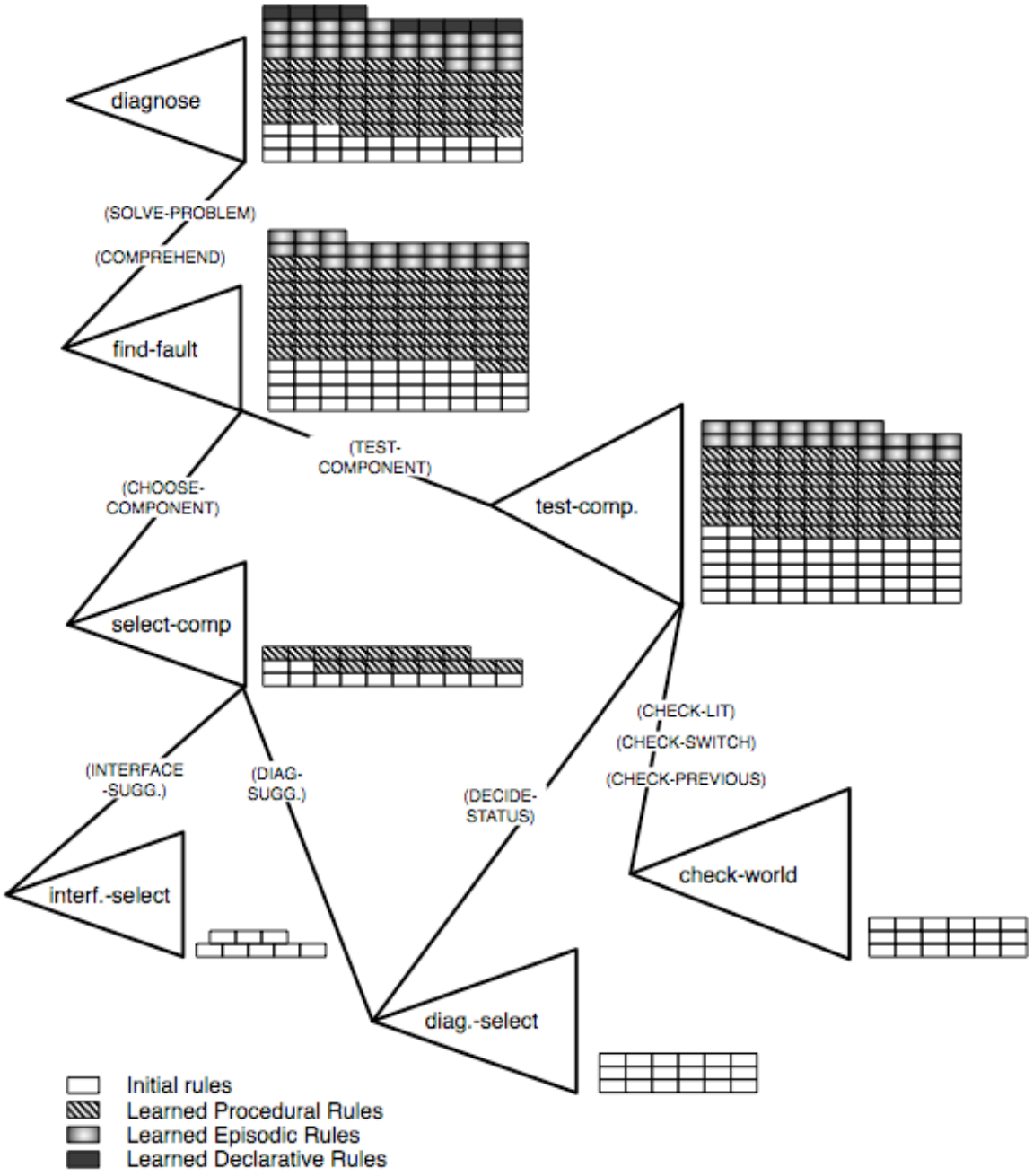


Fig. 10. Initial and learned rules organized by problem space.

Notes

[1] Although Anzai and Simon (1979) did look at detailed performance, it was for only one participant.

[2] Two example series have been created as pdf files and are included on the web site. Figures 8 and 9 show all of the series.

[3] The model, an executable version of Soar for the Macintosh, example traces, example stimulus sets (recreated as pdf files), and the data reported below are available at acs.ist.psu.edu/projects/diag

[4] This mechanism is called chunking in Soar, but is a type of procedural learning.

[5] We relabel these learned rules as LR-1 and so on; in the trace file they appear as Chunk-1 and so on.

[6] For simplicity and clarity we refer to Soar's decision cycle here as *model cycle* or just *cycle*.

[7] Altmann and Tafton (2002, Fig. 5) also found a fit this good for aggregate data.

[8] The differences between the predictions of the individual and global regression coefficients being used to generate predictions are small; both are included in the data analyses file on the web site.