Thema: Implementierung von schematischen Denkstrategien in einer höheren Programmiersprache:
Erweitern und Testen der vorhandenen Resultate durch Erfassen von zusätzlichen Daten und das Erstellen von weiteren Strategien

Masterarbeit

Im Studiengang Angewandte Informatik
der Fakultät Wirtschaftsinformatik und
Angewandte Informatik der
Otto-Friedrich-Universität Bamberg

Verfasser: Maik Friedrich

Themensteller: Prof. Dr. Ute Schmid

Abgabedatum: 30 April 2008

FACULTY OF INFORMATION SYSTEMS AND APPLIED COMPUTER SCIENCE
UNIVERSITY OF BAMBERG

Implementing diagrammatic reasoning strategies in a high level language: Extending and testing the existing model results by gathering additional data and creating additional strategies

Maik B. Friedrich

Masters Thesis

Applied Computer Science

30 April 2008

---

friedrichm@gmx.de

Supervisor: Prof. Dr. Ute Schmid from Otto Friedrich University, Bamberg

Co-supervisor: Prof. Dr. Frank E. Ritter from the Pennsylvania State University

Abstract

This thesis builds upon a cognitive model and a study by Ritter and Bibby (2008). This existing learning model called Diag was implemented in Soar 6 and is based on the idea that learning consists of procedural, declarative, and episodic learning. Despite the high correlation between observed and predicted performance there were participants whose behavior could not be predicted. Therefore, the main goal of this thesis is to understand the strategies humans use while doing a task of reasoning and learning, and to use this knowledge to predict human behavior. For this purpose, the model was reimplemented in a high level language, Herbal, and compiled into Soar 8. An additional user study was run to gather enough data for the analysis of different strategies and strategy shifts. It could be shown that participants created different strategies to solve the fault-finding task. Based on the observations, four additional strategies for the diagrammatic task were identified and implemented in strategy models. These strategy models were validated by comparing their predictions to the observations. As a result, it could be shown that participants not only created different strategies but some also shifted between strategies when solving the fault-finding task.

Table of Contents

1. INTRODUCTION

We are learning every day, for example, by reading the newspaper, watching a movie, or riding a bike. All these activities provide information that can be transformed into knowledge. We might not even notice how this happens, but we are able to recall the knowledge when needed. This process is part of learning which gives us the ability to create strategies. Strategy building is necessary for survival and, on a less fundamental level, to be a productive member of society. This thesis will give a more complete picture of strategy creation by analyzing a diagrammatic reasoning task done within a user study. The strategy research will be supported by a cognitive model called Diag which solves the task and predicts human performance.

The following introduction gives an overview of the motivation for this master's thesis. For this purpose, it is necessary to provide the reader with a short introduction to learning and an overview about the paper that was used as a basis for this masters thesis.

1.1. Learning

Learning is one of the most important abilities of humans. Of course, humans are not the only ones capable of learning, animals and some computer programs are able to learn as well. However, the human performance is still unmatched. The diversity of human learning can be seen in many fields e.g., learning in school, playing a game, or working at a job. Take, for an example, someone using software with an unknown interface. This user would gather new knowledge, based on the existing knowledge and depending on the individual learning ability. If, on the other hand, the interface designer had had information about the future users of his product, he could have created interfaces that are adapted to the users' abilities.

Learning itself can be described as conscious and unconscious, individual or collective acquisition of mental, physical, and social knowledge or abilities. This implies that people often cannot control what they are learning and that there is a difference between theoretical and practical knowledge. Another definition of learning is the process by which long-lasting changes occur in behavioral potential as a result of experience. This definition says that the information or strategies that are memorized and used to change the decision finding process can be declared as learning. A third, more task-oriented definition is "Learning is where performance changes with practice, typically getting faster, less effortful, and with fewer errors." (Anderson, 1995). This means that learning is the measurable improvement when a task is done repeatedly. All the different definitions above show the diversity of interpretations of what learning really is.

As there are different definitions for learning, there are different types of memory to store the learned knowledge. There are several possible ways to divide memory into memory types. For example, dividing the memory into implicit versus explicit memory parts. Another example is dividing the memory into declarative and procedural sections. Declarative memory describes acquiring facts e.g., refrigerators keep food cold and boots are made for walking. Procedural memory stores how to do something, e.g., how to play a game or how to cook a meal. Declarative and procedural memory can be used to define the following varieties of learning (Anderson, 1995):

1. *Cognitive stage* – understanding the basics of a task.
2. *Associative stage* – learning how to perform a task without errors.
3. *Autonomous stage* – optimizing the skills developed in the previous stage.

There are different theories about how learning is connected to problem solving and strategies. When someone is doing a task for the first time or even still while getting instructed, his mind tries to create a strategy for the task. This is called the skill acquisition phase and is followed by the conditioning phase. This means, after a person has understood how to do a task, the next step is to improve this knowledge if possible. This is especially correct for tasks which have to be done more than once. Measurable improvement of the abilities begins when doing the same task for the second time with the help of the previous strategy. This leads to an improvement in efficiency, time consumption, and errors made. The strategy is enhanced every time the task is performed. By measuring and drawing the improvements over several trials the curve shows a rapid improvement followed by lesser improvements with further practice (Newell & Rosenbloom, 1981; Ritter & Avraamides, 2001). This curve is called the learning curve (Figure 1) and the rate and shape of improvement is fairly common across tasks (Ritter & Schooler, 2001). For some tasks improvement continues for over 100.000 trials, this means that even after this amount of trials the performance gets improved. To advance learning or to help others to learn more quickly and efficiently, knowledge about how the mind creates, selects and improves strategies is essential.

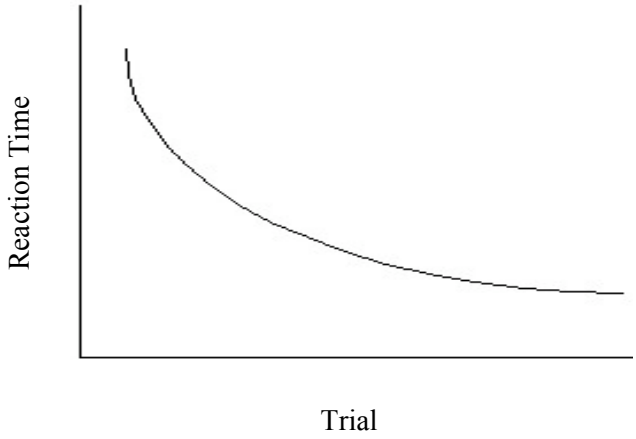


Figure 1. The learning curve in reference to time and productivity.

A better understanding of the learning process is useful for many scientific and practical fields, e.g., artificial intelligence, cognitive science, human resources, or everyone who is involved in education. For example, a teacher could optimize subject matters in a certain order and thereby help his students understand better. A large scale of improvement is possible if people learn in the correct order (Ritter, Nerb, O'Shea, & Lehtinen, 2007). Because the human model of learning seems to be effective, computer scientists could use this experience to develop better learning software and robots. Machine or computer interfaces could be designed for an easier understanding and remembering. This leads to a more efficient way of human computer interaction.

1.2. Learning models

A learning model is a concept that describes how humans study information and how they use that knowledge to solve problems. Since the beginning of cognitive science different models and mechanism have been developed to show how human learning occurs. A behaviorist view of learning supports a learning model based on the assumption that learning is a conditioned reflex that is caused by an appropriate stimulus. An artificial intelligence view declares that learning is a method of

changing a system. This change allows it to perform more efficiently on the repetition of the same task or on another task drawn from the same population. In education, learning models are attempts to describe how people and animals learn, thereby helping to understand the inherently complex process of learning. A cognitive view on learning mechanism was introduced by Chase and Simon (1973). They used the term chunk to indicate long-term memory structures that can be used as units of perception and meaning, and chunking as the learning mechanisms leading to the acquisition of these chunks.

Newell and Simon (1979) showed that the power of computers can be used to simulate complex scientific theories and predictions. This means the predictions that have been made by a computer simulation of a learning model can be used to determine whether that learning model is correct. Today there are several types of learning models implemented in cognitive architecture. According to a brief review by Ritter and Bibby (2008) these models include procedural knowledge compilations (Anderson, 2007; Gonzalez, Lerch, & Lebiere, 2003; Pavlik, 2007; Taatgen & Lee, 2003) and base level learning in ACT-R (Altmann & Trafton, 2002), connection strengthening in PDP models (O'Reilly & Munakata, 2000), rule creation from impasses using analogical reasoning (VanLehn & Jones, 1993), constraint elaborations (Ohlsson, 2007), and both declarative and procedural learning mechanisms (e.g., Feigenbaum & Simon, 1984; Rosenbloom & Newell, 1987).

One of the most advanced models of learning is the Diag model (Ritter & Bibby, 2008). This model is built on the idea that "procedural, declarative and episodic learning all are taking place within individual problem solving episodes" (Davis & Ritter, 1987). In Ritter and Bibby (2008) a fault-finding task was used to observe how learning takes place. User studies were run with this fault-finding task. Every participant had to learn instructions for the task. After that, the participants had to run 20 trials while their reaction time and choices were recorded. The Diag model was implemented to examine whether the model could predict the participants' learning process that took place while doing the task. The model was created in the Soar 6 cognitive architecture. The hierarchical structure of problem spaces and Soar's learning mechanism made the learning predictions possible. The Diag model is able to learn at a similar rate as most participants did. This enabled Ritter and Bibby (2008) to predict learning rates accurately.

The Diag paper opens up several opportunities for validation and extensions. For example, analyzing the different strategies could be a first extension. As shown by Ritter and Bibby (2008) not every participant used the same strategy to solve the fault-finding task. The Diag model supports a learning strategy that predicted 80% of the participants' behavior correctly. However, 20% of the participants were using a different strategy. Since Delaney, Reder, Staszewski, and Ritter (1998) requested that "Candidate architecture that seek to explain the acquisition of complex skills must accommodate strategy change and strategy variation and account for the independent speedup of each strategy", the analysis of different strategies would be a useful extension and therefore is a part of this thesis.

A problem in the study of complex problem solving, especially in a learning context, is the range of individual differences. In order to study the acquisition of complex skills, a good research strategy is based on the theory of individual differences. The creation of such a theory requires multiple participants and detailed data. Therefore, a second extension could be the collection of additional data. This should be combined with an expansion of data capturing methods, e.g., the use of an eye tracker to capture eye movement that can be used to trace the participants' steps. This might provide even more information about the actual thinking, problem solving, and learning processes. Because there were only two participants that did not fit the Diag predictions, additional data is needed to generate more strategies that do not correspond to the existing Diag strategy.

A third extension could be the reimplementation of the model in the latest version of Soar. The existing model was created in Soar 6 and the current version is Soar 8. This fact provides a good opportunity for analyzing if the Soar architecture has stayed the same. This is of particular interest since earlier version transitions in Soar "hardly corresponded to a progressive move from n^{th} to $(n+1)^{th}$" (Cooper & Shallice, 1995). Also, an additional Diag model in a high level language would be more flexible to create new models of learning with a different strategy, and this means a speedup.

1.3. Motivation

As demonstrated in section 1.2 there are a lot of possible research areas related to the Diag model (Ritter & Bibby, 2008). The goal of this thesis is to get a better understanding of the human ability to learn and create strategies. "New strategies frequently reduce the number of intermediate steps in solutions" (Delaney et al., 1998), and this does not necessarily imply an improvement. However, sometimes another point of view or a different strategy results in better and faster solutions. Research on the Diag model could be helpful in many different ways e.g., speeding up the learning process and supporting the development of new educational methods that are more related to the actual learning process. The information on how humans learn could give us the opportunity to reflect our own learning and strategy developing process more effectively. Improved interfaces that support the user in finding the optimal strategy or help to optimize the existing strategy for a task could be developed with a better knowledge about the human learning process. In this context, it is important to force users to contemplate about what they are doing and how they create their strategies. Therefore research in this area is important and useful for education, training science and applications.

The Diag model has shown to create promising predictions of how people learn. The model has the ability to learn procedural, declarative, and episodic knowledge. This means "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" (Ritter & Bibby, 2008). All these types of knowledge were used in several different models before but never combined in one model. This is what makes the Diag model a special, interesting, and valuable basis for this masters thesis.

To develop the additional Diag model the next step is to analyze the different strategies that were chosen by the participants and did not fit the model predictions. Assuming that the Diag learning model is correct, differing data could only derive from measuring problems (such as participants did not fully understand what to do, they had experiences in similar tasks, or were distracted) or different participant strategies. The existing Diag model is based on one strategy to solve the fault-finding task. This means by creating additional models with new strategies predictions that fit the data could be found. The following sub steps are necessary to do this.

The first sub step is to examine how people create their strategies when solving the fault-finding task. Ritter and Bibby (2008) published data from 10 participants and used an individual regression analysis for comparing the participants' performance to the Diag model predictions. The model was able to predict the performance of 8 participants with a correlation of 0.95. The model failed in predicting the performance of 2 participants. Ritter and Bibby (2008) argued that these participants could have used a different strategy as the Diag model, and thus the model was unable to predict their performances. To analyze different strategies for the fault-finding task, data from two individuals are not enough to generate a significant conclusion and therefore an additional user study needs to be run.

Within the additional study eye tracking data will be useful to see the order in which the participants used the interface information. In addition with the mouse movement data, this might give a more detailed view of how learning occurs. This way it could be shown how and when strategy shifts or improvements in form of learning take place. The changes to the original task should be kept to a minimum, however, some adjustments are necessary to make sure all participants understand the task completely and are motivated to generate their own strategy. Also, the originally used software to capture the participants' performance needs to be updated to a more current version.

Not only is additional user data needed, but also does the existing implementation of the Diag model need a renewal. The existing Diag model was written in Soar 6 which leads to difficulties while executing. The current Soar version is 8, and there have been some changes to the architecture. When changing the Diag model into different strategy versions parts of the model could be reused and expanded. The best way to avoid modifying Soar 6 directly is to implement the Diag model in a high level cognitive modeling language. This keeps the work for creating additional strategies to a minimum. Besides, Soar 6 is no longer supported by the Soar community which leads to problems when changing the model, e.g., no user manual for Soar 6 is available.

This masters thesis is structured according to the previously mentioned steps. First, a review of current types of problem solving strategies should give the reader some background knowledge and show how this thesis is related to current research in the field of cognitive science. Second, the Diag model is implemented in the high level behavior representation language Herbal, and thereby compiled into Soar 8. This way a comparison between the additional model and the existing model in terms of language (Soar 6 and 8) and behavior can be made. An important third step is the collection of additional data. The additional data facilitates the analysis of different strategies and strategy shifts in the fault-finding task. To prove that the Diag model is correct, the final step will be the development of different strategies based on the additional Diag model and the user study observations.

2. REVIEW OF CURRENT TYPES OF PROBLEM SOLVING

Human problem solving has been a central topic since the beginning of cognitive science. Newell, Shaw, and Simon (1958) outlined the standard theory of problem solving by focusing on how humans respond when confronted with unfamiliar tasks. Early work focused on the solution of problems such as puzzles e.g., the Tower of Hanoi or Missioners and Cannibals (Simon & Reed, 1976). Later research tried to explain cognition in semantic domains such as solving word problems in physics (Anderson, 2007; Bibby & Payne, 1993), as well as behavioral differences between task novices and experts (Bransford, Brown, & Cocking, 1999; Larkin, McDermott, Simon, & Simon, 1980).

Anderson (1996) defines problem solving as a goal-directed task. Within this task a sequence of operations has to be applied on a significant cognitive level. These criteria assure that a task is in fact a problem requiring a solving process. Otherwise, a task is not considered a problem solving task, e.g., knotting a tie when the person already knows how to do it. With no cognitive operators the task is only based on recalling information. The following sections give an overview of the current scientific background to this thesis. As a basis for chapter 5 and its results there is a review of the research on strategy shifts. For a better understanding of chapter 4, which outlines the learning process of the participants, a review on learning is provided. To support chapter 3, an overview of new techniques in cognitive modeling and an introductory view on Soar and Herbal is given.

2.1. Different strategies of problem solving

The first learning theories that were developed by Thorndike, Skinner, and Hull established learning as the central topic in psychology. New strategies and their construction are a part of learning, but historically they received relatively little attention. One reason is that the idea of strategies was uncommon because the stimulus-response connection was the main criteria for early analysis. Also, most of the tasks studied were extremely simple so that only easy strategies were needed. In addition, this kind of research is hard to create and requires more data. The transition away from behaviorism to cognitive psychology directed the focus for the analysis of learning to strategy research. This new approach appeared in the earliest publications of cognitive research, for example in Newell and Simon (1972).

Research indicates that experts use several key principles to solve tasks more effectively than novices. Experts pay more attention to meaningful patterns in the environment and recognize more possibilities to approach new situations while solving a problem. Their knowledge is set up to give a more precise image of the problem and cannot be reduced to sets of isolated facts or propositions but, instead, reflects contexts of applicability: that is, the knowledge is "conditioned" on a set of circumstances (Bransford et al., 1999). This means experts have learned to use more effective strategies than novices.

Before generating a strategy the acquisition of instructional knowledge on the task is necessary. After that the new problem solving skills can be developed. There are three different stages of skill acquisition that are described by Fitts (1964) and Anderson (1995). These stages are the cognitive stage, the associative stage, and the autonomous stage. They describe the process of transition from declarative to procedural knowledge. In the cognitive stage, the learner starts from examples or instructions of how to perform the task. In this stage the leaner can often be observed to rehearse the instructions. For example, learning how to use Microsoft Word: at first a teacher or tutorial explains what to do, while demonstrating. In the associative stage the skill makes a transition from a slow and deliberate use of knowledge to a direct realization of what to do. The process of solving the task

becomes more fluid, error free and verbalization of the skill drops out. For instance, underlining a word in Microsoft Word is done by double clicking on the word and then clicking on the underline button. In the autonomous stage the skill becomes continuously more automated and faster, and cognitive involvement is gradually reduced. Sometimes a person even loses the ability to verbally describe the skill or certain sub parts. In such a case, the skill totally becomes a matter of procedural memory. An example is when the user of Microsoft Word stops reading the button names and just clicks them.

Task solving strategies are created in the first stage of learning. They can evolve or be reconstructed in the second and third stage of learning. Some of the most detailed information on the structure of acquired memory skills and strategies has been collected in the studies of Staszewski (1988). He used multi digit multiplications without external memory aids to research the effects of extensive practice and different strategies. These strategies include the replacement of a series of computational steps with direct retrieval of results, the adoption of more efficient algorithms, and efficient memory management strategies. After hundreds of hours of practice on this task participants were able to increase their memory performance with the use of different strategies. The movement from one strategy to another, in this case from calculation strategy to retrieval strategy, is called strategy shift.

An example for a task that allows multiple strategies is underlining a word in Microsoft Word. The user has the possibility to use the help menu to get guided instructions to underline a word, to use the underline button on the menu, to use the font menu, or to use a keystroke. Another example for different strategies is the comparison of heuristics and algorithms in chess. Simon and Gilmartin (1973) pointed out that most chess experts discover combinations in chess games because their mental programs have a powerful selective heuristic for searching the next move. It is not the case that they memorize better or think faster than their opponents. For the selection of a chess move, the use of a heuristic is more time saving than the use of an algorithm that goes through every possible move. The use of the algorithm would take an enormous amount of time and memory; however, it would also definitely deliver the best move.

One famous problem for studying problem solving is the mathematical puzzle Tower of Hanoi. The basic version consists of three pegs and three disks of different sizes (Figure 2). The disks can slide onto any of the three pegs. The puzzle starts with the disks put on one peg in ascending order, the smallest on top. The goal of the puzzle is to move the entire stack to another peg, obeying the following rules:

- Only one disk may be moved at a time.
- Each move consists of taking the upper disk from one of the pegs and sliding it onto another peg, on top of the other disks that may already be present on that peg.
- No disk may be placed on top of a smaller disk.

Using a greater amount of disks can extend the puzzle and raise the complexity.

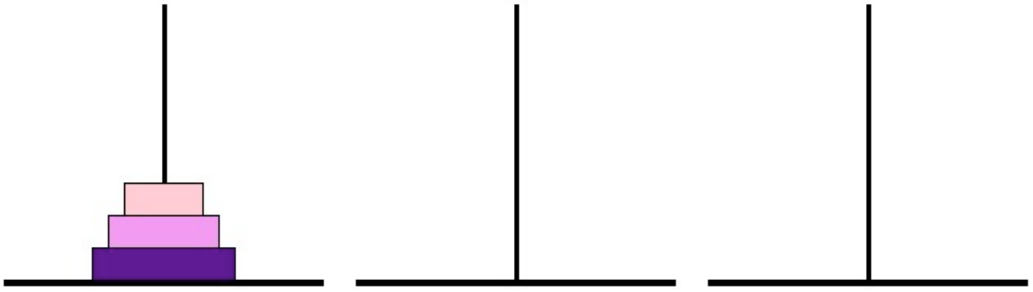


Figure 2. The experiment set-up which is used for the Tower of Hanoi.

The Tower of Hanoi is an example of a simple problem-solving task, but still complex enough to provide room for different strategies. Simon (1975) has analyzed and described strategies to solve the Tower of Hanoi. Three strategies from Simon are the Goal-Recursion-Strategy, the Perceptual-Strategy, and the Move-Pattern-Strategy.

The Goal-Recursion-Strategy is based on the goal of moving N disks from the start to the final peg. Since this goal cannot be reached in the beginning if $N > 1$, a sub goal is created. This sub goal is to move N-1 disks to the peg that is not the final or start peg. With each new sub goal the target peg changes. The procedure of creating sub goals is continued until the first sub goal can be reached, thus $N = 1$. This is an elegant solution because all operations are falling into place into the sub goal structure, but a large amount of memory is required to store the sub steps.

The Perceptual-Strategy uses the visible state of the puzzle to determine the next move and solve the problem. The first sub goal of this strategy is to find the direction the smallest disk will cycle, which depends on whether the number of disks in the puzzle is odd or even. Then the disk can be moved according to a simple pattern: the smallest disk is moved to the next peg in the cycle, and the next-smallest exposed disk is moved to the only available peg. This pair of moves is repeated until the puzzle is solved. This strategy can be performed without keeping track of the disks in working memory.

The Move-Pattern-Strategy is a complete retrieval strategy. Every disk move is known at the beginning of the problem solving task and gets recalled from memory. This makes this strategy the quickest solution for the problem, with no steps of calculation. The principal difficulty is the large number of steps that must be memorized and the fact that the steps are similar and hence susceptible to interference. Because the steps are different depending on the number of disks in the puzzle, this strategy cannot be generalized.

One of the first cognitive models that was compared to actual learning is that of Anzai and Simon (1979). They created an adaptive production system that was able to show the changes in strategy across four episodes of solving the Tower of Hanoi. Even though the model's behavior was not compared to subjects' behavior on an action-by-action level it produced the same strategy shifts as the subjects.

The word underlining task, the chess example, and the Tower of Hanoi show that there can be multiple strategies for the same problem solving task. The process of strategy development can be characterized, in part, by improvements in the adaptive use of iterative problem-solving strategies (Charness, 1991). For a given task an array of strategies is typically available. As demonstrated in the examples the strategies differ in the confidence in using a strategy, probability of producing the correct solution, the duration of problem solving processes, and the demands the strategy places on memory resources. The choice of a strategy is often based on the weight of these factors. A change in strategy occurs when the weight of these factors changes while doing a task. For a better understanding of the topic an overview of several publications on different strategies will be given.

To analyze strategy shifts Geary and Wiley (1991) chose a younger and an elderly aged group of participants. Both groups had to solve the same addition stimuli task (e.g., 9 + 6). The possible strategies for this task were separated into verbal counting, decomposition (e.g., 9 + 1 + 5) and memory retrieval. It was found that the elderly participants used the retrieval strategy more frequently than the young participants. They, on the other hand, relied on the decomposition and verbal counting strategies more frequently. Since the memory retrieval strategy is the fastest of all three strategies, the better performance of the elderly participants is based on the higher proportion of retrieval trials. The

fact that there were no group differences in the proportion of retrieval errors suggests a rather more peaked distribution of associations for elderly participants relative to the young participants. The data in fact showed that both groups had a rather stringent confidence criterion and that the elderly participants were able to compensate their slower information processing with the use of a more efficient strategy.

A similar study by Geary and Brown (1991) analyzed the same addition stimuli task but used gifted, normal, and mathematical disabled (MD) children as participants. Again, the different strategies were verbal counting, decomposition and memory retrieval. The data suggest that the gifted group appeared to use the retrieval strategy more often followed by the normal and MD groups. The gifted group also appeared to have a more stringent confidence criterion and was shown to switch to a more efficient strategy faster than the other groups. For the gifted group, the proportion of retrieval trials as well as the low error rate approached adult levels. The MD children appeared to have difficulties retrieving correct answers and therefore had to rely on the verbal counting strategy to solve most of the problems. The MD group also showed problems in choosing the correct strategy. The majority of the normal group fell in between the other two groups. Overall, this study showed that children with more basic knowledge about a task tend to choose the fastest and most efficient strategy.

Card, Moran, and Newell (1983) described strategy shifts while creating a model for text editing. They analyzed the behavior and strategies that participants used when making changes to a document. While doing the text editing tasks the participants tended to move the text to the upper 2/3 part of the monitor because the bottom was outside of comfortable view. Most participants had two strategies to move the view: POINT-WITHOUT-SCROLLING-METHOD and SCROLL-AND-POINT-METHOD. The experts had an additional strategy called JUMP-METHOD that allowed them to jump to a specific point in the text. For example, if the number of lines between two tasks was less or equal 4, the POINT-WITHOUT-SCROLLING-METHOD was often used. If the number of lines between two tasks was bigger than 16 all novice participants used the SCROLL-AND-POINT-METHOD, but the expert participants already switched to the JUMP-METHOD. Based on this Card et al. (1983) were able to show that strategy shifts can depend on an environment parameter.

Reder and Ritter (1992) used an arithmetic task in which participants had to choose rapidly between two strategies to solve two-digit times two-digit arithmetic problems (e.g., 44*18). Participants had 850 ms for the choice to either retrieve or calculate the answer. If they chose the retrieval strategy then they would have 1s to type the answer. If they used the calculation strategy then they would have essentially no time limit to calculate the answer. During the experiment, a number of specific arithmetic problems were repeated and as a result the participants' tendency to choose the retrieval strategy increased. A similar study with a multiplication task and the same result was performed by Siegler (1988). More examples of this type of strategy shifts are numerous, not only for arithmetic tasks. Studies on foreign vocabulary learning (Crutcher, 1989), spelling (Siegler, 1986), and the acquisition of linguistic rules (Bourne, Healy, Rickard, & Parker, 1997; Taatgen & Anderson, 2002) are all reflecting this type of strategy shift.

To analyze strategy shifts, the user study that was run for this thesis works with an experimental setup that was used by other studies as well. The setup consists of an interface, shown in Figure 3, which controls the electric circuit that is represented by a diagram, shown in Figure 4. The system consists of the power source (PS), two energy boosters (EB1 and EB2), accumulators (MA, SA1 and SA2), the laser bank (LB), the wiring, and the switches between them to route the energy. The switches on the interface represent the switches in the diagram and their arrows point the direction of the energy flow. The lights on the interface are illuminated if the component is not broken and energy is routed through

it. Scientists using this experimental setup were able to create a series of tasks, e.g., finding a broken component, finding the wrong switch position, or finding the broken light (Bibby & Payne, 1993).

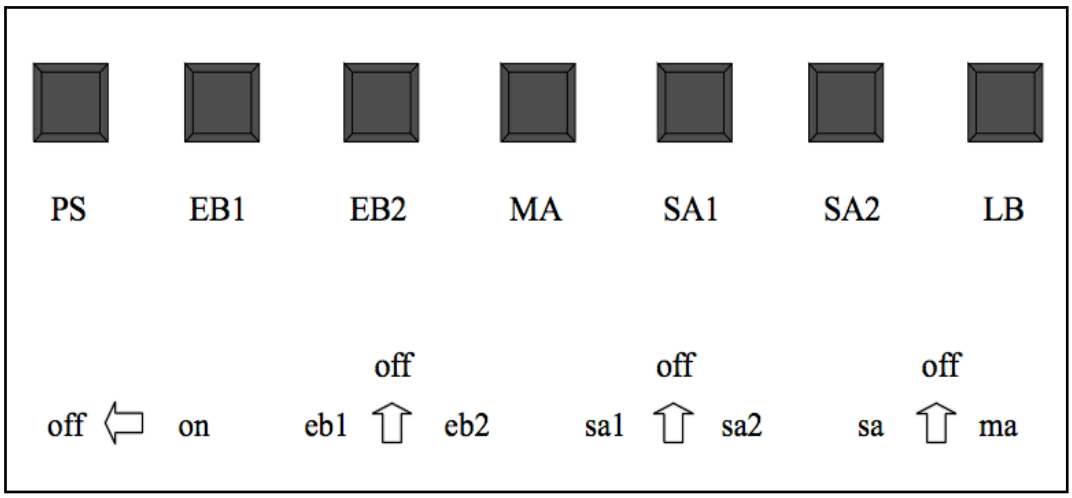


Figure 3. The fault-finding task interface (Ritter & Bibby, 2008).

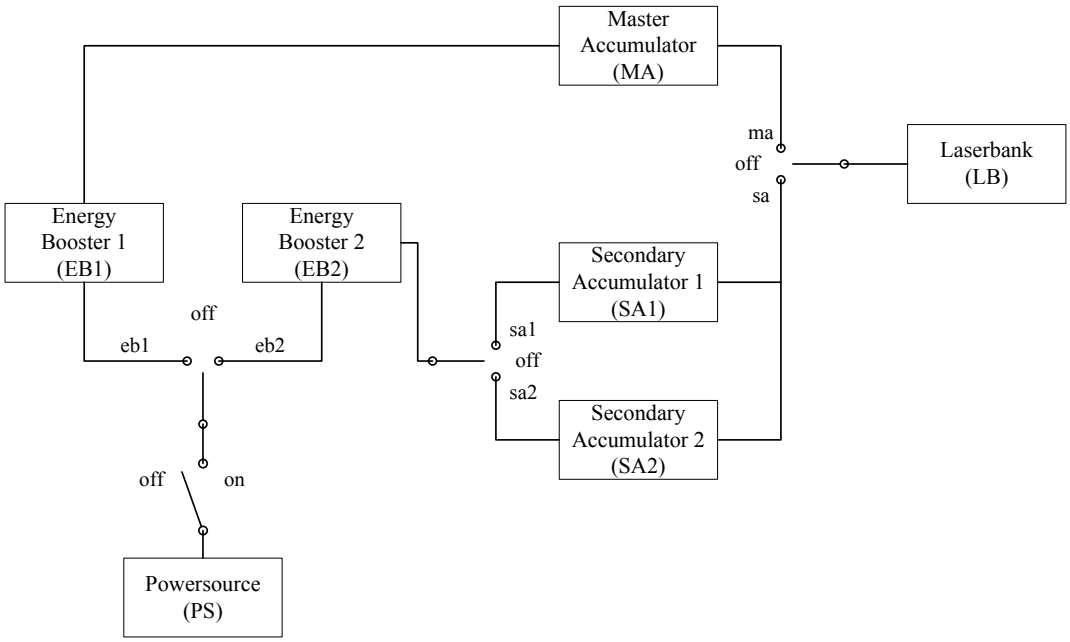


Figure 4. The fault-finding task circuit (Ritter & Bibby, 2008).

Previous work with this experimental setup had different research interests and results. Kieras and Bovair (1984) published summaries of behavior studies on a series of related tasks and the role of representing background knowledge at the beginning of the task. Kieras (1988) and Kieras and Bovair (1984) showed that knowledge of the system structure, the underlying principles that control the system, and how to perform a task are the instructions that need to be conveyed to the participants. This means if a task has different instructions which transfer the same knowledge in different ways, one instruction could be more helpful than the others. Bibby and Payne (1996) and Bibby and Reichgelt (1993) ran experiments using different sets of instructions to analyze the different participant behavior for non-faulted circuits. They chose experiments to examine whether their

definition of internalization combined with the ACT-R theory of skill acquisition (Anderson, 1983) provides good results.

Bibby and Reichgelt (1993) also used this experimental setup to perform a fault-finding task and develop an executive model. The person solving the fault-finding task is given the information that the energy has already been routed by positioning the switches and that one component within the routed way is broken. The goal is to find the broken component by analyzing the information given in the interface. The executive model has the ability to perform multiple tasks but no learning mechanism is described.

Ritter and Bibby (2008) also use the fault-finding task for research purposes. They developed a model called Diag. It was implemented in Soar 6 and therefore can use a learning mechanism that is similar to human learning. The model reaches its goal state by using the interface combined with the diagram knowledge. The strategy that Diag uses to solve the fault-finding task starts with the first component in the circuit. If the component light is off, PS is the broken part, otherwise Diag moves to the next light in the row. If EB1 is on, the model moves to the next light. If EB1 is off, Diag looks at the second switch to see if the energy is routed through EB1. If the second switch is set to EB1, the broken part is identified, otherwise Diag moves to the next light. This pattern continues until the broken component is found. To solve this task Diag uses 7 problem spaces and 22 operators.

The Diag model was able to predict an average proportion of 79% variability in problem-solving time (Ritter & Bibby, 2008). This performance could be improved even more by using reflective learning.

2.2. Reflection in problem solving

"Reflection is the process of stepping back from an experience to ponder, carefully and persistently, its meaning to the self through the development of inferences; learning is the creation of meaning from past or current events that serves as a guide for future behavior" (Daudelin, 1994, p. 39). This definition describes reflection within the learning process as the contemplation about past experience to affect and understand future experience, or an internal dialogue that individuals develop to help creating skills that predict outcomes and monitor comprehension.

Reflection consists of self-reaction and self-judgment. Self-judgment involves monitoring the own performance while solving a problem and assessing the solution. This personal performance gets self-evaluated by a standard or a goal. The assessment of errors while solving a problem is especially effective in sustaining motivation during periods of solution (Zimmerman, 1996), because strategy attributes sustain perceptions of effectiveness until all possible strategies have been tested. Reflective problem solvers are assessing errors as variables to continuously improve performance. This effect is increased with episodic problem solving because forgetting is minimized (Davidson & Sternberg, 2003).

The increased effects of reflective learning in episodic problem solving have several reasons, e. g. a better planning algorithm, faster motional skills, or the effect of attributed errors. Lüdtke et al. (2006) developed a Prolog model that used step-skipping to increase performance while maneuvering a plane through difficult air tasks. The model was able to simulate that increased step-skipping also increases the error rate. In a different experiment, using artificial algebra with adults, Blessing and Anderson (1996) were able to show that improved performance did not only come from step-skipping but also from decreased initial planning and the use of multistep planning.

The existing Diag model is implemented in Soar and therefore uses chunking as its most basic learning mechanism. Chunking acquires rules from goal-based experience and stores this knowledge in new rules[1]. By using chunking the Diag model is able to increase its performance while solving the fault-finding task. The learned rules are useful for implementing procedural knowledge directly, and implement learning effects directly as additions to the different model states. Other learned rules represent the use of proposing operators, which implement shifts of external attention. After every task the number of rules that are needed and the new rules that are created additionally to solve the task decreases, while the total set of rules increases.

The research on strategies and especially strategy shifts is a complex area with no binding definitions. In this context, strategy shifts are defined as the change of a complete strategy while doing an episodic task. This does not include changes within a strategy for optimization, e.g., changing from one step counting to two step counting while using the verbal strategy for an addition task. Choosing a different strategy while doing an episodic task depends strongly on a self-reflection after every task cycle (Zimmerman, 1996).

Ritter and Bibby (2008) ran user studies and collected the data of ten participants. Eight participants fit the Diag model predictions well. Therefore Ritter and Bibby (2008) developed several ideas why two participants did not match. Since the number of mistakes was similar for all participants, a possible explanation was that these participants used different strategies than the Diag model. Another explanation was that the participants did not need the usual time to solve a task which can be ascribed to better self-reflection or faster cognitive reflexes. Ritter and Bibby (1997) claimed that the detailed comparison between the participants' and model's behavior suggests that the model is not learning in all the ways the participants did. In addition, the model could include reflect on its performances or changes its time course of processing.

2.3. New techniques for modeling

Nearly all programming languages are equally powerful in the sense of being Turing equivalent, but most programmers do not care about being formally definable. They select a programming language based on its abilities. For example, when language A has an operator for removing spaces from strings and language B does not, that does not make A more powerful because a subroutine to do the same process in B can probably be written. However, if A supports recursion and B does not, this lack cannot be fixed by writing library functions, and therefore programmers would tend to select language A.

Besides the categories of functionality, programming languages can be classified by their level of abstractness. Languages in one area fall along a continuum of abstractness, from the less abstract (low-level) to the most abstract languages (high-level), which themselves vary in power. For example a database developer would no longer choose a machine language to program databases. He would choose a database language because it makes implementing and managing databases easier. This leads to the general principle: when a programmer has a choice of several programming languages on the abstractness continuum, the most abstract one is probably the best (Graham, 2003).

[1] Soar uses this term for rule learning. Most psychologists consider chunks as an appropriate model of declarative memory.

Progress equals a further development of a state, mostly in the positive sense. From a perspective of cognitive architectures, there are two sources of individual differences: architectural differences and knowledge differences (Taatgen, 1999). Structural design differences vary in the cognitive architecture itself, e.g., how global parameters can be defined, or working memory can be influenced. Differences in knowledge are based on the idea that people use different problem solving strategies. In terms of a cognitive model, this means individualized models have different initial contents of declarative and procedural knowledge and different structures which determine the order to apply this knowledge.

The abstractness continuum for cognitive modeling languages is defined by behavior representation. Since the beginning of cognitive modeling numerous programming languages have been developed and used. It is possible to classify these languages according to their style of programming. The two styles are rule-based programming and higher-level description programming. However, in the context of this thesis, the rule-based languages will be considered as low-level behavior representation languages because they are the less abstract languages for cognitive modeling. On the other hand, the higher-level description programming languages will be considered as high-level behavior representation language because they are more abstract.

Low-level behavior languages are Jess, ACT-R, and Soar. These three languages provide a clear description of the state-of-the-art of low-level behavior representation. Jess (Java Expert System Shell) is a complete expert system shell that was developed by Ernest Friedman-Hill. A complete introduction to Jess and its functionality can be found at Friedman-Hill (2003). ACT-R was developed by John Anderson at Carnegie Mellon University. ACT-R is based on the theory of cognition called Adaptive Control of Thought-Rational (Anderson, 1993), which implies that human behavior is rational. ACT-R not only is a rule-based behavior programming language but also a cognitive architecture. This means it was created to implement a unified theory of cognition, and provides the infrastructure to create models that are based on the supporting theory by Newell (1990).

Because the existing Diag model is written in Soar, a detailed description for Soar is provided in section 2.3.1.

High-level behavior representation languages are modeling on abstract levels to simplify the encoding of behavior. This leads to a more direct representation of behavior. RAP, JACK, TAQL, High Level Symbolic Representation (HLSR), GOMS-based language, and Herbal are the state-of-the-art of high-level behavior representation languages. As reviewed by Ritter et al. (2006), all these languages use a high level of abstractness that makes it easier for cognitive developers to implement their models.

Reactive Action Packages (RAPs) was designed to specify tasks and to plan in a way flexible enough to deal with the uncertainty of an agent's interaction with a complex and unpredictable environment (Firby, 1989). JACK is based on the concept of the Belief-Desire-Intention (BDI) framework. Within this framework, an agent is defined by the parameters belief, goal, intention, and plan. The agent sets off with a goal and its beliefs about the world. In the next step the agent creates intentions which is followed by the execution of certain plans (Evertsz, Ritter, Russell, & Shepherdson, in press). Task Acquisition Language (TAQL) (Yost, 1993) is a high-level language designed to map directly to the Problem Space Computational Model (Lehman, Laird, & Rosenbloom, 1996) and to compile into Soar productions. The High Level Symbolic Representation (HLSR) project is aimed at creating a formal language that encompasses a wide variety of modeling tasks using a variety of cognitive architectures. Importantly, HLSR strives to make it easier to create models by providing high-level language support for common modeling problems.

G2A is a GOMS-based high level representation language that uses the higher-level Goals, Operators, Methods, and Selection Rules (GOMS) to create ACT-R models (St. Amant, Freed, & Ritter, 2005). GOMS itself was developed in 1983 by Card et al. (1983). After these initial publications whole families of GOMS-based languages evolved. GOMS based languages can be used to model the error free performance of skilled users. Advantages of using GOMS are its simplicity, abstractness, and the direct mapping to the task being modeled. The use of GOMS by G2A allows modelers to create behavior using an explicit representation of a theory, and this representation is automatically translated, using a compiler, into the low-level rules required by ACT-R.

Since the additional Diag model is written in Herbal, a detail description of Herbal is given in section 2.3.2.

2.3.1. Soar

Soar was developed by Allen Newell, John Laird, and Paul Rosenbloom at the beginning of the 1980's. In this section, a review of Soar in general and the relevant aspects of the architecture that influence and support Diag's learning functions and problem solving is given. Like ACT-R, Soar not only is a rule-based behavior programming language but also a cognitive architecture (Lehman et al., 1996). Soar was developed for the purpose of constructing general intelligent systems. It has been in use since 1983, and has evolved to the current version 8.6.3 which was released in October 2006. The cornerstones of Soar's design are:

1. The number of distinct architectural mechanisms is minimized. In Soar there is a single representation of permanent knowledge (productions), a single representation of temporary knowledge (objects with attributes and values), a single mechanism for generating goals (automatic subgoaling), and a single learning mechanism (chunking).

2. All decisions are made through the combination of relevant knowledge at runtime. In Soar every decision is based on the current interpretation of sensory data and any relevant knowledge retrieved from permanent memory. Decisions are never precompiled into uninterruptible sequences (Laird & Congdon, 2005).

The structure of Soar is oriented at Newell's Unified Theory of Cognition (UTC) (Newell, 1990). Newell's UTC tries to give a cognitive plausible explanation of how intelligent organisms flexibly react to stimuli from the environment, how they acquire goals rationally and exhibit goal-directed behavior, as well as how they represent knowledge and learn. Soar provides the user with the necessary tools to implement Newell's UTC. This implementation is realized implicitly by using rules. This rule-based concept makes finding the theoretical elements of Newell's UTC difficult when examining Soar code. This also leads to a low-level of abstraction and not to the explicitness that the theory proposes.

Soar does not implement the Problem Space Computational Model (PSCM) directly, but supports it (Lehman et al., 1996; Newell, Yost, Laird, Rosenbloom, & Altmann, 1991).The PSCM is based on the primitive acts that are performed while using problem spaces to achieve a goal. Problem spaces are common collections of states, sets of states, goals, and a set of valid operators which contain the constraints under which to apply themselves. The top-level goal is to transform the initial state into the goal state. This goal is reached by applying operators. A state consists of a set of literals that describe the knowledge of the agent and the present model of the environment. New goals or sub problem spaces are generated when there is no available operator that would move the agent closer to its goal state. The behavior of the Soar agent is defined as the movement through a problem space.

Soar supports working memory (WM) and long term memory (LTM). Figure 5 shows the memory structure of Soar. The LTM consists of procedural, semantic, and episodic knowledge. Changes to this knowledge influence the WM to apply operators that move the current state towards the goal state. This process is repeated in regular intervals until the current state equals the goal state. In Soar an interval is defined as a decision cycle. This decision cycle consists of two phases, elaboration followed by decision. During the elaboration phase all productions which match the current working memory are applied in parallel until no more productions match. The decision phase evaluates every preference put into the working memory and chooses the next problem space, state, operator, or goal to apply.

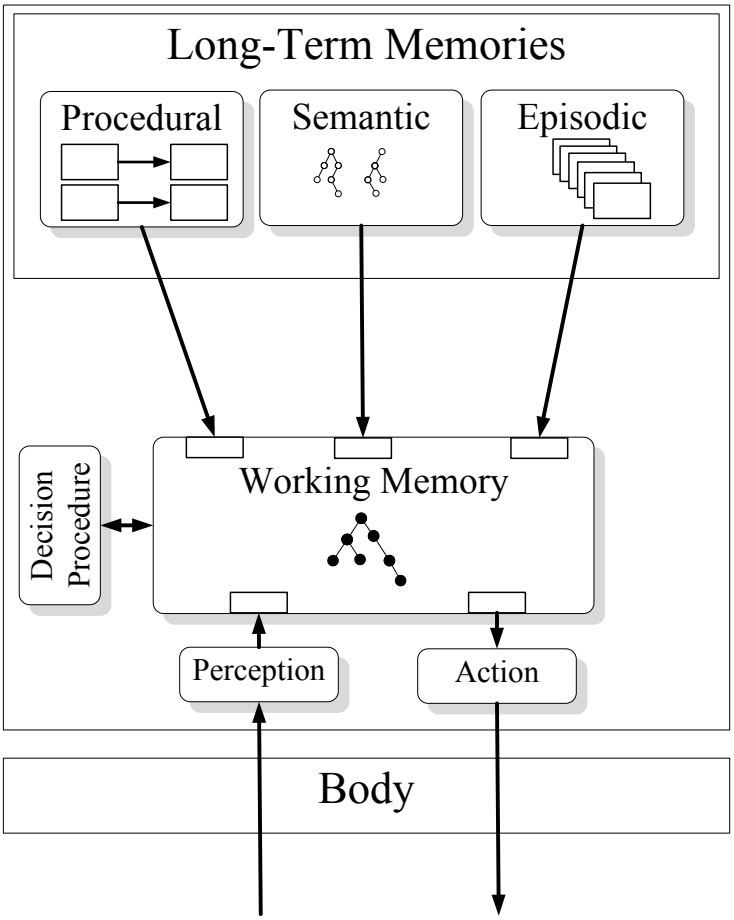


Figure 5. Soar Working Memory structure (Laird, 2003)

Soar has a learning mechanism called chunking that allows agents to learn while performing a task. Production rules that are created by chunking are called Chunks. These Chunks memorize the initial state of a sub problem space and the results it generates. When a Chunk is created, it is added to Soar's production memory.

A requirement for the creation of Chunks (learned rules) is that the sub problem space passes back changes to the super problem space, but can only learn from the bottom problem space. When a similar situation occurs, Soar uses the Chunk to avoid running the sub problem space again. Since most Soar agents continuously use sub problem spaces and return results to their parent problem spaces, Chunks are typically created continuously as Soar runs.

Soar implements the PSCM implicitly using rules. This leads to differences between the theory behind the PSCM and its implementation in Soar. As mentioned earlier, the Soar programming language is a low-level behavior representation language. Consequently, every modeler who wants to encode a task in Soar has to map the problem description into the PSCM, followed by mapping the PSCM

description into the abstract rule-based model, and then finally into the actual syntax. Behavior mapping requires a set of complex skills which a modeler has to acquire before modeling in Soar.

In 1993, the Diag model was implemented by Sam Marshall using the Soar 6 architecture (Laird, Newell, & Rosenbloom, 1987; Newell, 1990). On the PSCM level, Diag consisted and still consists of 7 problem spaces that are hierarchically ordered. Within these problem spaces 20 operators (Figure 6) are grouped to control the switching between different problem spaces and therefore perform the problem-solving task. These problem spaces and operators are implemented through 170 Soar rules. Since the implementation of Diag, changes have been made to the Soar architecture. Those changes and their influence on the Diag model will be discussed in section 3.2.

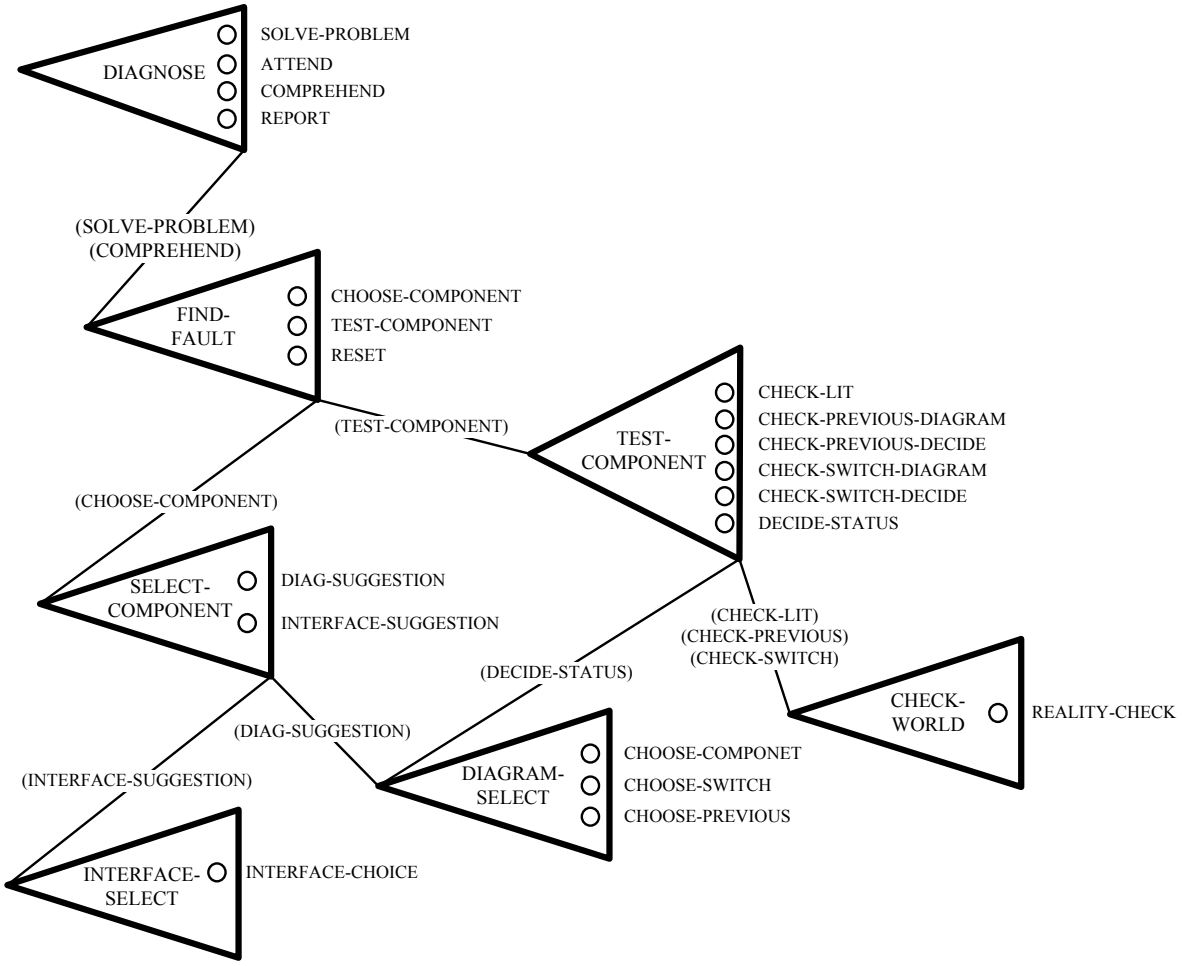


Figure 6. The problem space structure of the existing Diag model (Ritter & Bibby, 2008).

2.3.2. Herbal

Herbal is a high level behavior representation language. Herbal was developed by the Applied Cognitive Science Laboratory located in the College of Information Sciences and Technology at the Pennsylvania State University (Cohen, Ritter, & Haynes, 2005). The Herbal development started in 2002 and currently supports the creation of models for the Java Expert System Shell (Jess) and the Soar Cognitive Architecture.

A main goal of developing Herbal was the reduction of complexity in cognitive modeling. Herbal's high-level language is based on the PSCM, allowing for models to be created at a level of abstraction

above the standard production level. Herbal allows the cognitive modeler to focus on the architectural aspects of their cognitive agent while the details of programming are managed by the Herbal compiler. This way Herbal represents a step towards the development of tools which support modelers of intelligent agents and cognitive behavior. Herbal is designed to create agents which have the ability to explain themselves. This is possible through the formalization of the programming process in Herbal with the help of an explicit ontology of classes that represent concepts of the PSCM. These classes are the basis for the Herbal structure (Cohen, Ritter, & Bhandarkar, 2007).

The Herbal environment consists of the high-level language, a graphical editor, a Soar debugger, and a compiler. The design was influenced by usability studies done in human behavior modeling and artificial intelligence courses. In order to support beginner and advanced users the Herbal environment has two interfaces, the graphical editor, also called Herbal GUI Editor, and the modifiable XML files. Both interfaces are consistent which means changes entered in the GUI Editor are automatically transferred to the XML file and the other way around. The graphical interface is recommended for Herbal beginners and makes the introduction to the system easy. Furthermore, a Herbal tutorial helps starters even if they do not posses any knowledge about cognitive modeling (Cohen et al., 2007).

Eclipse is an open-source software framework written primarily in Java. In its default form it is an Integrated Development Environment (IDE) for Java developers. Users can extend its capabilities by developing toolkits for other programming languages, and can write and contribute their own plug-in modules. The Herbal GUI Editor is implemented as such a plug-in. This minimizes problems with Herbal for users who are familiar with Eclipse. Figure 7 shows the Herbal GUI Editor after installing the plug-in. The plug-in comes with an automatic compiler that transfers every change in the Herbal GUI Editor or the XML model files directly into the output files. A Soar remote debugger is also part of the plug-in. It allows debugging the running Soar kernel.

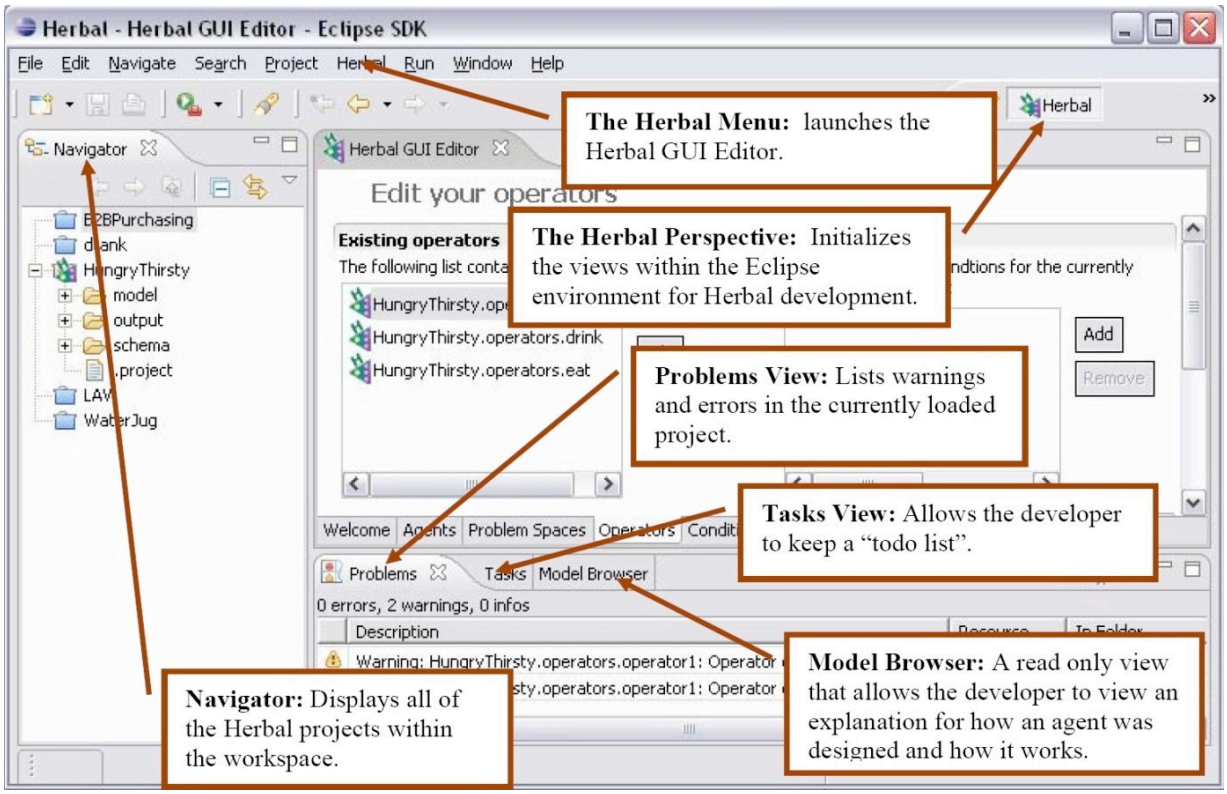


Figure 7. Graphical Herbal Interface within Eclipse (Cohen et al., 2007).

The Herbal compiler is the main component of Herbal and has several functionalities. First, as mentioned before, when Herbal is used as Eclipse plug-in the compiler works automatically whenever

a change is made to one of the interfaces. Secondly, the compiler can be used without Eclipse to compile Herbal projects directly by using a command shell. The compiler also uses the design rationale in Herbal to create expressions of self explanation while the Soar model is running. By doing so, Herbal models can give explanations about how and why they use a component. The Herbal debugger is also aware of this design rationale, the PSCM structure of the Herbal model, and, therefore, creates an output that supports the Herbal developer.

As mentioned before, Herbal is based on classes that represent concepts of the PSCM including models, states, operators, elaborations, conditions, actions, and working memory. The Herbal developer uses the entities agents, problem spaces, operators, conditions, actions, and types to combine the present concepts of the PSCM into a structure or rather a high level behavior representation language. A schematic view of the Herbal structure plus the responsible XML files is given in Figure 8. The elements of Herbal depend on a separate XML and XSL file. This modular architecture simplifies the handling of Herbal projects and is helpful while debugging. The output files are created by the compiler and this way independently from the XML files.

Figure 8 shows that the agent is the highest entity in Herbal and consist of a collection of problem spaces. The highest problem space has the global goal. The agent only exists to reach this goal. The entity problem space can consist of several sub problem spaces which are also goal driven. These are local sub goals that serve the top problem space. The procedural knowledge in every problem space is represented by its operators. An operator consists of actions and conditions. Conditions are patterns that match states in the working memory. Actions lead to the creation, change, or removal of facts in the working memory which results in the agent moving closer to the global goal. When created, an operator defines the conditions in the working memory that trigger an action to working memory. The entity Types defines the instantiation of working memory elements. Types are the equivalent to data types in a traditional programming language.

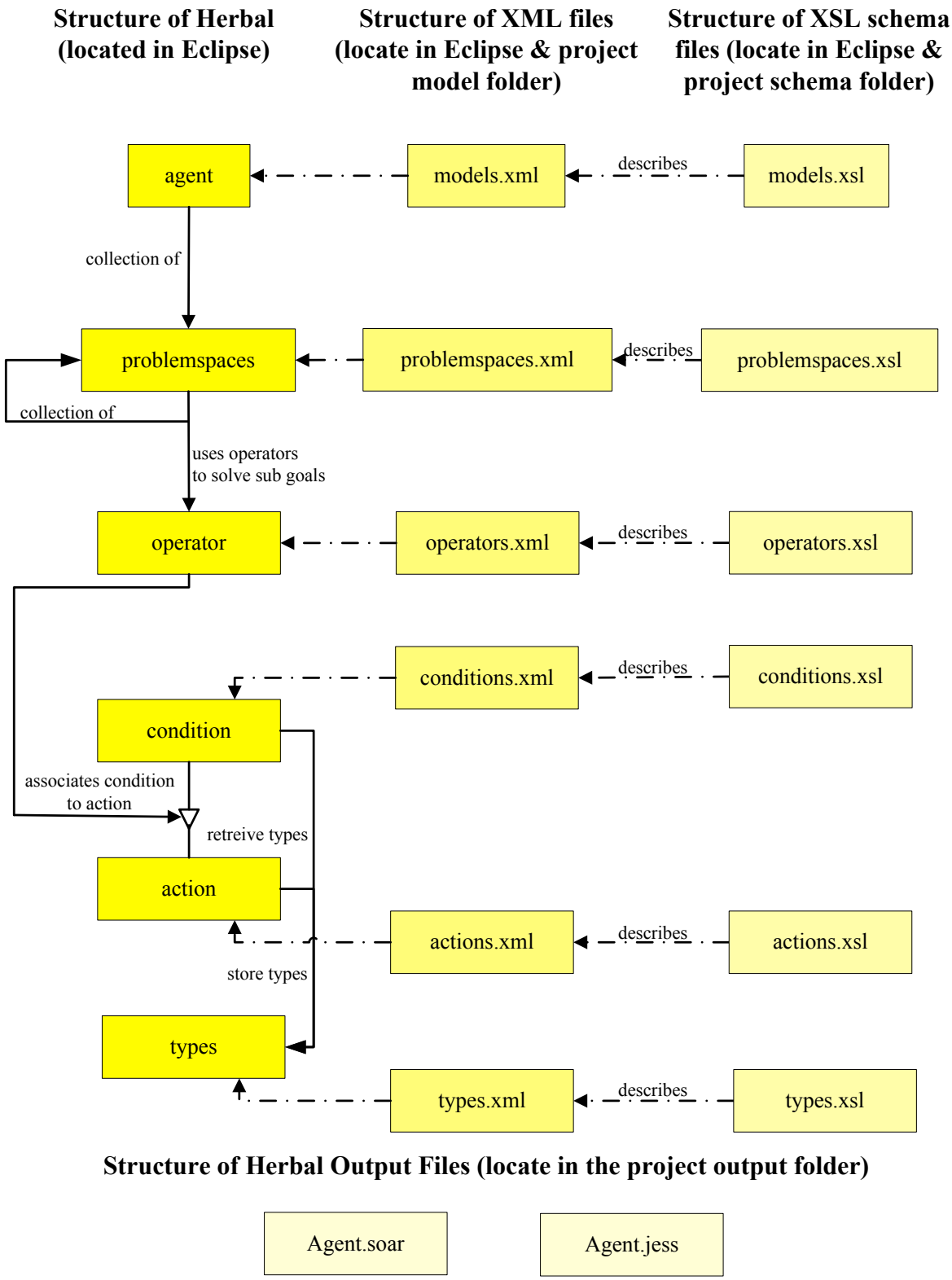


Figure 8. The Herbal file structure and its association with the Herbal elements (Friedrich, Cohen, & Ritter, 2007).

2.4. Suggestions

This review suggests several aspects which should be considered to get additional results and an expended knowledge about strategies and strategy shifts. The Diag model is ideal for this purpose because it has a high potential for analyzing human behavior. A key aspect is the influence that different strategies have on the model predictions. Research on the different strategies could be supported by collecting additional user data. This might result in a better understanding of the problem solving processes on a cognitive level, give a better view on the occurrence of learning in a particular task, and perhaps lead to suggestions for how learning might happen in other tasks.

2.4.1. More strategies and distributions of strategies

The problems and tasks described in section 2.1 are used to study human behavior, human problem solving, and strategy shifts. As a necessary basis for this research possible strategies have to be investigated and categorized to evaluate their influence on the task solving process. For example, participants solving the Tower of Hanoi used the Goal-Recursion-Strategy, the Perceptual-Strategy, the Move-Pattern-Strategy, or a mixture of these strategies. Therefore, all three strategies can have an influence on the participants' performance. Another example was described by Card et al. (1983) where participants tended to use a monitor scrolling strategy depending on their knowledge and the number of lines. Card et al. (1983) were able to show that for a range of lines that need to be scrolled down, e. g. between 1 and 16, the participants tended to prefer one strategy to the others. The results fluctuated when expert and beginner results were compared.

The Diag model is based on one strategy to solve the fault-finding task and predict human behavior. However, there were some examples of participants using a different strategy. The analysis of the existing results as described in section 2.2 shows that some model predictions do not match the participant performance. In order to research on the different strategies presented in section 2.1, the next step towards a learning Diag model is the analysis of possible strategies for the fault-finding task. The existing Diag model strategy should be modified and implemented in different model versions.

2.4.2. More user studies to collect additional data

This review also suggests running more user studies. As mentioned in section 2.4.1, Ritter and Bibby (2008) studied 10 subjects, 8 of whom fit well to the predictions. It was suggested that the two non fitting participants could have used a different strategy. The main goals of this thesis are the implementation and analysis of strategies in addition to the existing strategy that is used by the Diag model. The first step to accomplish this goal would be to analyze the two participants that did not fit. Unfortunately, two participants are not enough to create significant scientific conclusions about behavior. Therefore additional data is needed.

When running user studies to gather additional data to an existing set of data, an important thing is to keep the experimental setup and the process of running the study as close to the original study as possible. This requires the use of comparable introduction material, a similar environment for solving the task, and the same verbal instructions as given to the original participant. While the methodology for the study is available, the research assistant who ran the study is not available. If there were subtle, or informal features used in this study, they cannot be reproduced. Therefore, some adaptations to the

process of data gathering might be inevitable, e.g., switching to new hardware for representing the environment.

Before performing a user study that is looking for strategy differences, it is difficult to predict the number of participants and data granularity necessary for a significant result. A larger number of participants provide better insight into the learning process. In addition, with detailed enough data, it is possible to retrace a larger number of intermediate order effects. In this way Ritter et al.(2007) were able to show that participants starting off with different strategies could reach the same level of performance. However, there is a trade-off. A larger data collection automatically means a more complex analysis of the data which might lead to a more general examination. This can result in generalized results and interesting individual interaction might be ignored. Assuming the percentage of participants that did not use the Diag strategy in the existing user study (Ritter & Bibby, 2008) is 20%, 30 to 40 participants might be the optimum number of participants to minimize the trade-off in an additional study. This number would provide around 6 to 8 participants who use a different strategy.

2.4.3. Better understanding of the Diag model

The Diag model was able to predict human behavior in problem solving and therefore is a promising candidate for further research on strategy and strategy shifts (section 2.4.1). The model is based on the PSCM, but implemented in an architecture that only indirectly supports the PSCM. This implementation consists of rules that draw a concrete picture of the model but in a low-level behavior language. Therefore, understanding the existing model (170 rules) by looking at the source code is difficult even for an experienced Soar programmer.

Diag should be implemented in a high-level cognitive representation language because:

- Creating an implementation in a language that is based on the PSCM allows a more abstract view of the Diag source code.
- A high-level language Diag model would be more flexible for changes and tests of new designs.
- Soar 6 is no longer supported by the Soar community and therefore renewing the Diag model is necessary to make it accessible to other users and projects.
- Diag would be available for a larger community because the exchange of models in high-level languages is easier.

Herbal is an appropriate tool for this task because it covers all of the issues above.

3. REWRITING THE DIAG MODEL IN HERBAL

As mentioned before, rewriting the Diag model in Herbal is a necessary and important step in order to analyze the strategies used to solve the Diag task. The Herbal model makes future adaptations to the model, e.g., changing the strategy, easier and allows a more flexible soar code management. This chapter gives a detailed description of the rewritten Herbal model and its distinctions compared to the existing Soar 6 model. Also the predicted data from the existing model and additional model are compared to identify possible predicting differences.

3.1. The Herbal Diag model

The Herbal Diag model is supposed to behave like the existing model. Therefore the structure of the existing model (Figure 6) was used as a blueprint to model the same structure in Herbal. This section covers all important additional model components and their functionality. An overview of all problem spaces and their operators is shown in Figure 9. Compared to the existing model only five new operators were implemented and their most common use is to interact with the Java environment and to create a more detailed picture of the program structure.

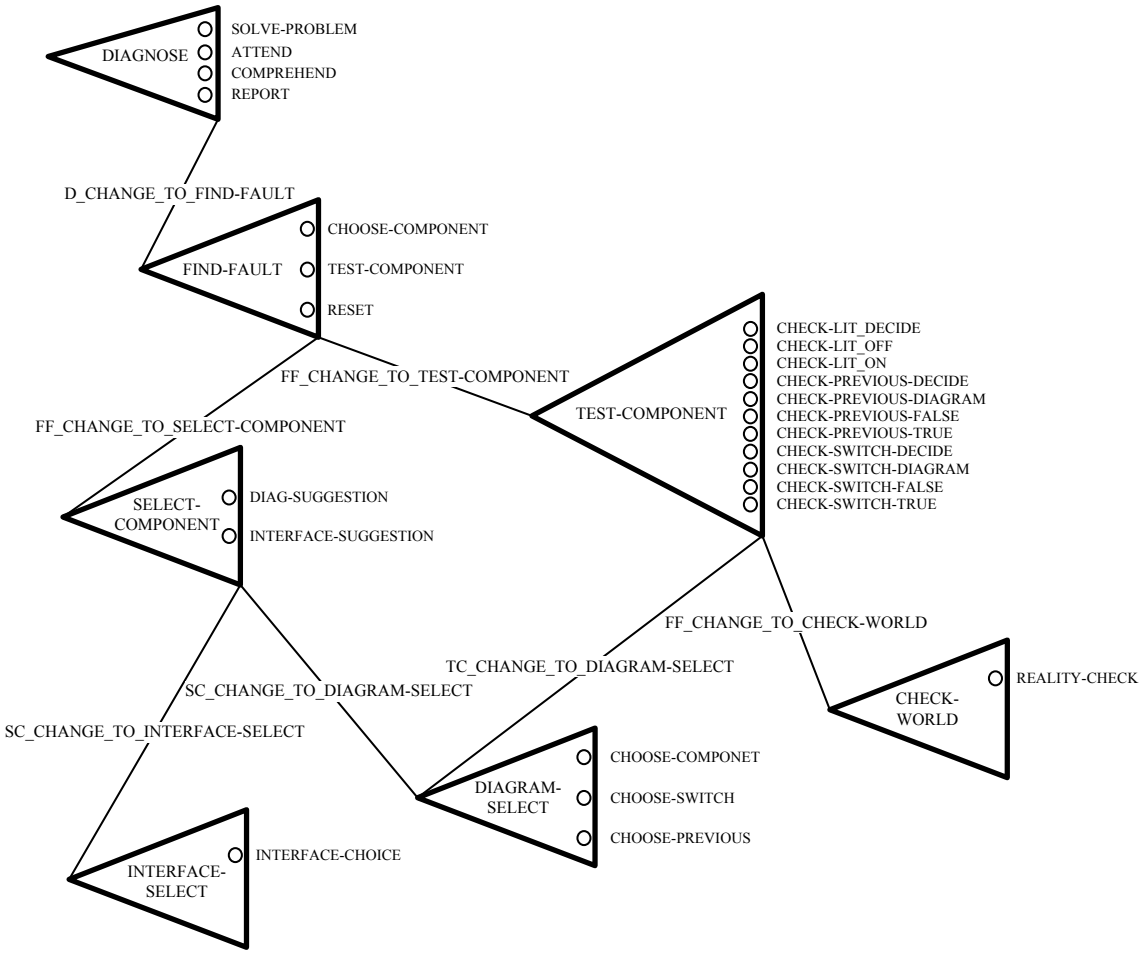


Figure 9. Problem space structure of the additional Diag Model.

3.1.1. Stages of model development

While creating the additional model in Herbal several problems emerged. The first was that Soar 6, which is the developer environment for the existing model, is no longer supported on most operating systems. Therefore just a printed trace provided by Ritter and Bibby (2008) allowed a view on the running model. It was also difficult to find a Soar 6 manual that explains the existing model trace. A source of information was the Soar 8 manual that explains the Soar 8 debugger output and therefore indirectly the Soar 6 output.

Within the process of implementing the additional Diag model in Herbal, three different kinds of models were developed. The three types differ in the position of the working memory and how the communication with the environment supports the problem solving. The first type was implemented with completely separate working memory. Operators were used to request information from the environment and copy the answers into the model's working memory. This type of Herbal modeling is difficult to manage because the memory is used inefficiently, e.g., the complete memory is stored in the environment and the model at the same time. This can lead to problems, e.g., if the model tries to copy information into the working memory that is already in there, this information is removed from the memory. This happens because Soar removed the information from the Soar kernel, and then does not allow the same information to enter the working memory until the decision cycle is finished.

The second type was implemented with a mixture of working memory. Some information about the problem state was managed directly in the working memory and other parts such as knowledge bases were implemented in input link elements. Operators were still used to request information and change working memory elements. They were not used for copying information from the input link into working memory. This leads to operators that try to change the working memory, and request knowledge at the same time. This can lead to problems with the Soar learning mechanism, e.g., if an operator changes the sub problem space to its goal state and causes a change in the working memory, Soar tends to create over specific productions that will never apply again.

The third type was implemented with no internal working memory elements and the complete knowledge is represented by input link elements. The operators send requests via the output link to change the environment. This results for example in putting a requested component into the TO-CHECK working memory element. After the action has been executed, the input link gets updated with the new state of the environment. This type of Herbal model only has request operators and does not need multi operators. It also has problems with the Soar learning mechanism because Soar tends to learn to fire two requests at the same time. This leads to a bottleneck at the environment input link. In this case the environment was advised to execute one action from the model and remove the other from the input link without execution.

With the help of these three types four different models were created. Only two of them were able to learn without making any mistakes. From these two only one learns in a way that is similar to the existing model behavior.

3.1.2. The structure and components

As described in section 2.3.2, a Herbal model consists of one agent with problem spaces, operators, conditions, actions, and types. In order to describe the additional Diag model the agent with its problem spaces and operators will be explained. The model assumes that the basics of how to solve the fault-finding task is understood, e.g., which component to choose first or how to test a component. The

model begins the problem solving at the same level the participants were when they were able to pass the diagrammatic training test but had not achieved any significant increase in performance.

Diag finds the fault by starting at the power source and testing along the circuit structure. Knowledge of the structure is represented in three ways, knowledge about the interface, knowledge about the diagram, and knowledge about the current state of the environment, e.g., light EB2 is off. The interface and diagram knowledge are based on recalled memory. The current state of knowledge is obtained from a simple model of vision supplied by the environment and represented as a declarative structure. A human problem solver collects this knowledge through natural language processing and processing of graphical representations. Diag does not explain how knowledge is acquired. The order in which the problem spaces and operators are applied is based on a common strategy developed by Bibby and Payne (1996).

Figure 9 shows the additional Diag model divided in its problem spaces. Every problem space has a sub goal state and operators to reach these goals. Within every problem space the available operators are applied to change the internal state until the internal state equals the sub goal state. The problem spaces are connected by conditions which manage the transitions from one problem space to another. The additional Diag model has seven problem spaces that are linked by seven conditions. This hierarchical structure implemented in Herbal and supported by Soar is what enables the model to learn while solving the problem. 25 operators are performing the fault-finding task. They are based on 45 conditions and 35 actions. Diag has 9 types to manipulate the working memory. This includes input and output link elements. Herbal compiles the Diag model to a Soar 8 file with 72 rules. The file is loaded into the Soar kernel and used together with the environment.

The DIAGNOSE problem space is used as a top problem space and therefore initializes the task and reports the result back to the environment. It contains four operators. When started, the model applies the SOLVE-PROBLEM operator. This operator defines the goals for the DIAGNOSE problem space and therefore starts the problem solving process. After the SOLVE-PROBLEM operator was applied no operator can fire. At this point Soar creates an impass that changes the model into FIND-FAULT problem space. The ATTEND operator is an interface that manages 70% of the output messages from the model to the Java environment. This is necessary for the request of knowledge about the current state of the environment, e.g., checking whether PS is lit up or SWITCH2 is pointed to EB1. In order to avoid that two contradictory things are learned at once, a common method used in Soar is accessing the external world within the top problem space (Newell, 1990). Therefore, the model has to use the ATTEND operator to attend to the environment. The COMPREHEND operator is used to establish the model state after using the ATTEND operator. The REPORT operator is used to send the information about the faulty component to the environment and therefore restarts the model on the next problem in a series of faults.

The FIND-FAULT problem space manages the actual fault finding process and consists of three operators. The CHOOSE-COMPONENT operator sets up the problem state for the transition into the SELECT-COMPONENT problem space. The TEST-COMPONENT operator chooses the next component in the circuit that needs to be checked. This decision is based on the goal state of the SELECT-COMPONENT problem space. The TEST-COMPONENT operator also sets up the TEST-COMPONENT problem space to test whether the current component is faulty or not. If the current component is not faulty, working memory needs to be reset. This is realized by the RESET operator. After resetting the FAULT-FINDING problem space the CHOOSE-COMPONENT operator is applied again to select the next component for testing.

The SELECT-COMPONENT problem space manages the requests to the interface and diagram knowledge base. It has the operators DIAG-SUGGESTION and INTERFACE-SUGGESTION. They are used in a random order. Both knowledge requests set up the sub problem spaces to gather the required knowledge.

The INTERFACE-SELECTION problem space accesses the interface knowledge. This knowledge specifies the order of the components on the interface. This depends on the LAST-CHECKED component. For example, if PS was checked last then EB1 is next. The INTERFACE-CHOICE operator is collecting the information from the input link and therefore INTERFACE-SELECTION reaches the goal state. A person solving the fault finding task has the interface knowledge available through the analysis of the interface.

The DIAG-SELECTION problem space accesses the diagram knowledge. This knowledge can be described as information about the circuit structure. The CHOOSE-COMPONENT operator is similar to the INTERFACE-CHOICE operator. It collects the information about the next component based on the LAST-CHECKED component on the input link, e.g., SA1 and SA2 when EB2 is the previously checked component. The second operator CHOOSE-SWITCH is used to access knowledge about which switch is responsible for which component, e.g., SWITCH2 is responsible for EB1 and EB2. The third operator CHOOSE-PREVIOUS is needed to access knowledge about which component lies before another component in the circuit, e.g., EB1 lies before MA. CHOOSE-SWITCH and CHOOSE-PREVIOUS are used by the TEST-COMPONENT problem space to collect information for checking a component. A human problem solver has to use the declarative memory to access this information.

The TEST-COMPONENT problem space determines whether a component is faulty or not. Therefore it has to check:

- if the component is lit or not;
- if the switch responsible for the component is connected to the component;
- if the previous component is faulty or not.

When all three tests classify the component as faulty, the faulty component is found. As soon as one test results in not faulty, the component is not faulty. Therefore the goal states for this problem space are: component is faulty or component is not faulty. TEST-COMPONENT has 11 different operators which can be categorized in CHECK-LIT (CHECK-LIT-DECIDE, CHECK-LIT-ON, CHECK-LIT-OFF), CHECK-PREVIOUS (CHECK-PREVIOUS-DIAGRAM, CHECK-PREVIOUS-DECIDE, CHECK-PREVIOUS-ON, CHECK-PREVIOUS-OFF), CHECK-SWITCH (CHECK-SWITCH-DIAGRAM, and CHECK-SWITCH-DECIDE, CHECK-SWITCH-TRUE, CHECK-SWITCH-FALSE).

The CHECK-LIT-DECIDE operator decides whether to access the state of the environment and therefore sets up the CHECK-WORLD problem space. The CHECK-WORLD problem space sets up the transition to the DIAGNOSE problem space. As mentioned earlier, the ATTEND operator sets the information to make the decision if the component is lit up or not. Returning to the TEST-COMPONENT problem space, the CHECK-LIT-ON or CHECK-LIT-OFF operator fires to decide whether to perform another test or to set the component as not faulty.

The CHECK-PREVIOUS-DIAGRAM operator first sets up the DIAG-SELECTION problem space to collect information about the previous component. After collecting the necessary information,

CHECK-PREVIOUS-DECIDE sets up the CHECK-WORLD problem space. The same order of operators described for CHECK-LIT-DECIDE is used to gather the information needed for CHECK-PREVIOUS-ON or CHECK-PREVIOUS-OFF to fire.

The CHECK-SWITCH-DIAGRAM operator first sets up the DIAG-SELECTION problem space to detect the name of the switch which is responsible for the component. After this information has been collected, CHECK-SWITCH-DECIDE sets up the CHECK-WORLD problem space. The same order of operators described for CHECK-LIT-DECIDE is used to gather the information needed for CHECK-SWITCH-TRUE or CHECK-SWITCH-FALSE to fire.

If a check turns out positive and the currently checked component is not faulty, the TESTCOMPONENT problem space reaches a goal state. The model changes back to FINDFAULT and fires the RESET operator. If all checks turn out negative, the currently checked component is the faulty one. After reaching the goal state the model changes back to the DIAGNOSE problem space and fires the REPORT operator. The strategy defines that the TEST-COMPONENT problem space always checks the light first because it is the quickest and easiest check. The next check is either PREVIOUS or SWITCH. By analyzing the source code of the existing model it was found that the decision about what check to run next was done randomly because both are equal in complexity. This can lead to different results when the model is run again on the same set of problems. The additional model was implemented the same way.

The CHECK-WORLD problem space has one operator called CHECK-REALITY. This operator is used to change the model state to the DIAGNOSE problem space in order to access the environment or the problem knowledge respectively. CHECK-REALITY is doing this by resolving the impass for the DIAGNOSE problem space with a request message and thus reaching the goal state of CHECK-WORLD. This leads to a transition from CHECK-WORLD to DIAGNOSE problem space.

In addition to the problem space description Table 1 shows all operators and their description.

Table 1. The operators in the additional Diag model with description of their functionality and problem space.

Operator	Description	Problem space
SOLVE-PROBLEM	Operators for solving the task.	DIAGNOSE
ATTEND	Operator for implements the pseudo-I/O of the model	DIAGNOSE
COMPREHEND	Operator for changing back to problem space FIND-FAULT after ATTEND.	DIAGNOSE
REPORT	Operator to return the faulty part to the environment.	DIAGNOSE
CHOOSE-COMPONENT	Operator to start the search for a component to check.	FIND-FAULT
TEST-COMPONENT	Operator to start the test for a component.	FIND-FAULT
RESET	Operator to reset the working memory to rerun the FIND-FAULT problem space	FIND-FAULT
DIAG-SUGGESTION	Supports the diagram-knowledge	SELECT-COMPONENT
INTERFACE-SUGGESTION	Supports the interface knowledge	SELECT-COMPONENT
DIAG-CHOICE	Operator to select a component to check on the basis of the diagram knowledge.	DIAG-SUGGESTION
CHOOSE-SWITCH	Operator to select a switch on basis of diagram.	DIAG-SUGGESTION
CHOOSE-PREVIOUS	Operator to select a previous component on basis of diagram.	DIAG-SUGGESTION

INTERFACE-CHOICE	Operator to select a component to check on the basis of the interface knowledge.	INTERFACE-SUGGESTION
CHECK-LIT (3 Operators)	Operator to check a selected component if the is lit up or not	TEST-COMPONENT
CHECK-SWITCH (4 Operators)	Operator to check a switch if it is pointed to the current component.	TEST-COMPONENT
CHECK-PREVIOUS (4 Operators)	Operator to check if the previous component is broken or not.	TEST-COMPONENT
CHECK-REALITY	For transition to the DIAGNOSE problem space to access the environment.	CHECK-WORLD

The behavior of the additional Diag model is best explained by an example. Running the model with the power source as the faulty component would lead to the following order of operators and learned rules (LR). For a better orientation Figure 10 shows the order of the operators.

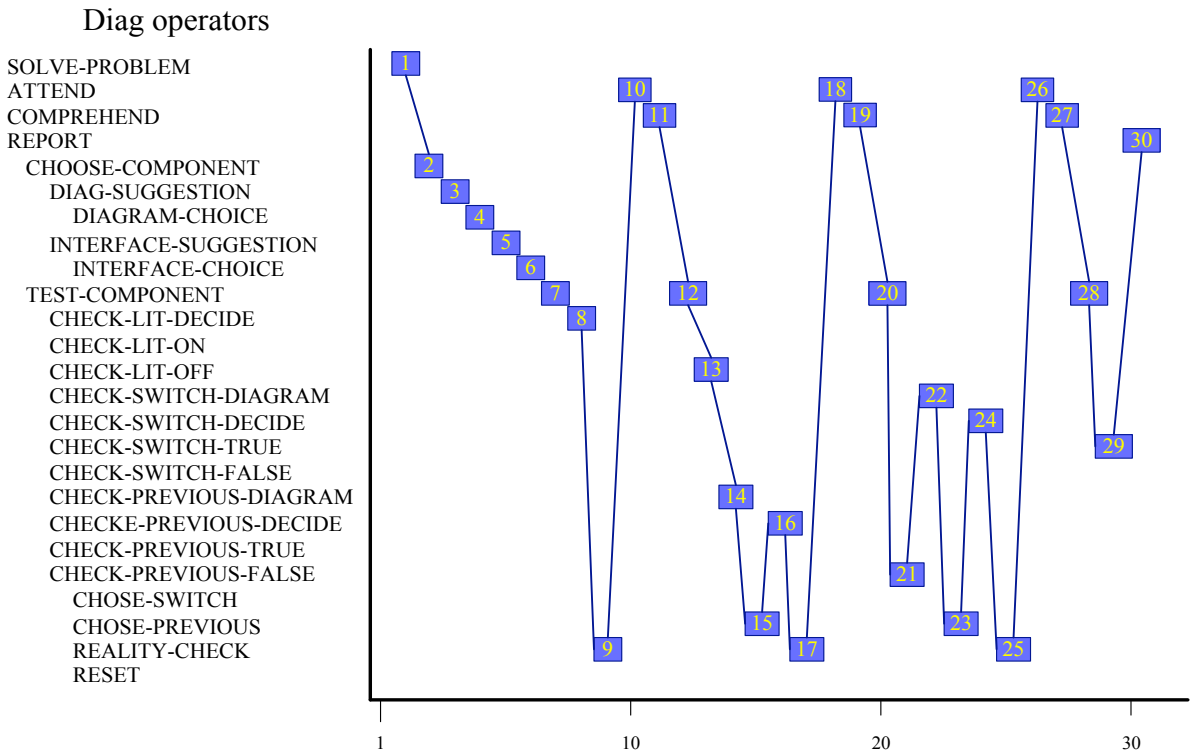


Figure 10. Additional Diag operators and their order when solving the fault-finding problem.

1. SOLVE-PROBLEM operator fires and changes the model state in order to change the problem space. No other operator fires and an impass is created when changing to the FIND-FAULT problem space.

2. CHOOSE-COMPONENT operator fires and changes the model state in order to change the problem space. No other operator fires and an impass is created when changing to the SELECT-COMPONENT problem space.

3. DIAGRAM-SUGGESTION operator fires and changes the model state in order to change the problem space. No other operator fires and an impass is created when changing to the DIAGRAM-SUGGESTION problem space.

4. DIAGRAM-CHOICE operator fires and changes the model state diagram knowledge depending on the previously checked component. Because the previously checked component is NONE the diagram knowledge is changed to PS (LR1).

5. INTERFACE-SELECTION operator fires and changes the model state in order to change the problem space. No other operator fires and an impass is created when changing to the INTERFACE-SELECTION problem space.

6. INTERFACE-CHOICE operator fires and changes the model state interface knowledge which depends on the previously checked component. Because the previously checked component is NONE the interface knowledge is changed to PS (LR2).

7. TEST-COMPONENT operator fires and changes the model state in order to change the problem space. No other operator fires and an impass is created when changing to the TEST-COMPONENT problem space.

8. CHECK-LIT-DECIDE operator fires and changes the model state in order to change the problem space. No other operator fires and an impass is created when changing to the CHECK-WORLD problem space.

9. REALITY-CHECK operator fires and changes the model state to DIAGNOSE problem space (LR3).

10. ATTEND operator fires and changes the model state to get access to the requested environment knowledge. In this case the value of the light is OFF.

11. COMPREHEND operator fires and changes the model state in order to change the problem space. No other operator fires and an impass is created when changing to the FIND-FAULT problem space.

12. TEST-COMPONENT operator fires again to reset the model state in order to change to the TEST-COMPONENT problem space.

13. CHECK-LIT-OFF operator fires because PS is not lit up.

14. CHECK-PREVIOUS-DIAGRAM operator fires and changes the model state in order to change the problem space. No other operator fires and an impass is created when changing to the DIAGRAM-SELECTION problem space.

15. CHOOSE-PREVIOUS operator fires and changes the model state diagram knowledge depending on the component that needs to be checked. As this component is PS, there is no previous operator.

16. CHECK-PREVIOUS-DECIDE operator fires and changes the model state in order to change the problem space (LR4). The TEST-COMPONENT problem space checks if the component previous to PS is faulty. No other operator fires and an impass is created when changing to the CHECK-WORLD problem space.

17. REALITY-CHECK operator fires in order to change to the DIAGNOSE problem space. The problem space reaches its goal state and the model changes to the DIAGNOSE problem space (LR5).

18. ATTEND operator fires and changes the model state to get access to the requested problem knowledge.

19. COMPREHEND operator fires and changes the model state in order to change to the FIND-FAULT problem space. No other operator fires and an impass is created when changing to the FIND-FAULT problem space.

20. TEST-COMPONENT operator fires again to reset the model state in order to change to the TEST-COMPONENT problem space.

21. CHECK-PREVIOUS-FALSE operator fires because the previous component is not faulty.

22. CHECK-SWITCH-DIAGRAM operator fires and changes the model state in order to change the problem space. No other operator fires and an impass is created when changing to the DIAGRAM-SELECTION problem space.

23. CHOOSE-SWITCH operator fires and checks the diagram knowledge depending on the component that is currently checked. Since this component is PS the switch to be checked is SWITCH1 (LR6).

24. CHECK-SWITCH-DECIDE operator fires and changes the model state in order to change the problem space. No other operator fires and an impass is created when changing to the CHECK-WORLD problem space.

25. REALITY-CHECK operator fires and changes the model state in order to change to the DIAGNOSE problem space (LR7).

26. ATTEND operator fires and changes the model state to get access to the requested problem knowledge. This knowledge contains the information that SWITCH1 points to ON.

27. COMPREHEND operator fires and changes the model state in order to change to the FIND-FAULT problem space. No other operator fires and an impass is created when changing to the FIND-FAULT problem space.

28. TEST-COMPONENT operator fires again to reset the model state in order to change to the TEST-COMPONENT problem space.

29. CHECK-SWITCH-TRUE operator fires because the switch position of SWITCH1 is on, which means the component is faulty. At this point all three tests are negative. Therefore it has been demonstrated that PS is the faulty component.

30. REPORT operator fires and informs the environment that PS is the broken part.

This example provides a complete view on the model's behavior which consists of firing operators, learning new rules (Table 2), and solving the fault finding task. Compared to the existing model (Ritter & Bibby, 2008) the behavior seems to differ. However, it has to be acknowledged that these differences are based on the replacement of general operators with more specific ones, which is explained next.

Table 2. The learned rules for solving the fault finding task the first time with PS as the faulty component.

Learned rules	If	Then
LR1	The model changes to the DIAGRAM-SUGGESTION problem space and the previous component is NONE	Change the value of DIAGRAMSUGGESTION to PS
LR2	The model changes to the INTERFACE-SUGGESTION problem space and the previous component is NONE	Change the value of INTERFACESUGGESION to PS
LR3	The model changes to the CHECK-WORLD problem space with PS as the component to check and CHECK-LIT as the type of test	Set up model state to change to DIAGNOSE problem space and fire the ATTEND operator.
LR4	The model changes to the DIAGRAM-SUGGESTION problem space to request the diagram knowledge and gather the information which component is previous to PS	Change the value of DIAGRAMSUGGESTION to NONE
LR5	The model changes to the CHECK-WORLD problem space with NONE as the component to check and CHECK-PREV as the type of test	Set up model state to change to DIAGNOSE problem space and fire the ATTEND operator.
LR6	The model changes to the DIAGRAM-SUGGESTION problem space to request the diagram knowledge and gather the information which switch is connected to PS	Change the value of DIAGRAMSUGGESTION to SWITCH1
LR7	The model changes to the CHECK-WORLD problem space with SWITCH as the component to check and CHECK-SWITCH as the type of test	Set up model state to change to DIAGNOSE problem space and fire the ATTEND operator.

3.1.3. Adaptations to the existing model structure

The most important aspect of reimplementation was to make as few changes as possible to the structure of the existing Diag model. Since the existing Diag model was modeled directly in Soar 6 and the additional implementation was made in Herbal, changes to the model structure were inevitable while adapting to the new language and environment. As mentioned before the developing environment that is used to run the Soar kernel was implemented in Java 1.5.

The first adaptation was splitting several operators into more detailed operators. This change did not have any influence on the problem space structure of the model but it reduced the complexity of the model components. This might not be true for Soar itself, but Herbal is designed to manage different operators that do one thing, consisting of conditions and actions. An example of this splitting process is the CHECK-LIT operator. In the existing model this operator had 2 functions. The first function sets up the CHECK-WORLD problem space and starts a request to the environment. The second function decides if the light is on or off, and then sets up the DECIDE-STATUS operator for the next step. Within the additional Diag model the CHECK-LIT operator is split into CHECK-LIT-DECIDE (setting up the CHECK-WORLD problem space), CHECK-LIT-OFF (changing to a different test because this one was negative), and CHECK-LIT-ON (changing to the FIND-FAULT problem space and resetting the testing procedure). These 3 operators are implementing the same functionality as the

CHECK-LIT operator in the existing model. This adaptation was necessary because the creation of operators with conditions that depend on the model state is not possible in Herbal. This means a new operator is created whenever the same action is needed in different stages of the Herbal model.

The next adaptation was the replacement of the problem space transitioning operator by Herbal conditions. This means that the existing model uses operators to change between problem spaces. This is not possible in Herbal because the transition between two problem spaces needs to be defined as a Herbal condition. This condition and an initial action are combined to an initializing operator. Every Herbal problem space has such an initializing operator. It can also be described as a transition operator with no link between condition and action. For example, in the existing model the transition between SELECT-COMPONENT problem space and INTERFACE-SELECTION problem space was managed by the INTERFACE-SUGGESTION operator. The Herbal model uses the INTERFACE-SUGGESTION operator to set up the model state for the CHANGE_TO_INTERFACE-SELECTION condition. Because the INTERFACE-SELECTION problem space has no initial action, the transition operator between SELECT-COMPONENT and INTERFACE-SELECTION has only one condition. This adaptation was necessary because Herbal does not allow modeling a transition between two problem spaces in any other way.

The last adaptation is that the model state is managed by the environment. The Herbal model operators do not influence the model state directly. They use the output link to send action to the environment to change the model state. This means that the diagram, interface, and problem knowledge is managed by the Java environment. The Java environment allows the implementation of all necessary data structures. This adaptation was necessary because Herbal only supports the memory types string, integer, and boolean.

The influence of the changes to the existing Diag model structure on the learning ability is described in section 3.2.1.

3.1.4. Suggestion for Herbal and Herbal users

Herbal is a valuable language to model Soar and Jess code on the PSCM level. The adaptation (section 3.1.3) and the process of reimplementation of the Diag model led to a list of interesting suggestions for future versions of Herbal.

The first suggestion is an automatic function for creating standard conditions and actions. This could be implemented as an extra window in the graphical Herbal type view. Whenever a new type is implemented in Herbal a popup menu could allow the automatic creation of standard conditions or actions for this type. For example, after creating a new Herbal type called LIST this menu would allow creating LIST-DOES-EXIST or LIST-DOES-NOT-EXIST conditions that are automatically added to the Herbal condition view. These conditions will be needed to check whether LIST exists in the working memory. Since these conditions are needed repeatedly when working with Herbal, the programmer safes the time when creating simple conditions.

The next suggestion is a copy and paste function for agents, problem spaces, operators, conditions, and actions within the graphical environment. The only way to make a copy of one of these elements is by editing the xml files. This might not be a problem for small models but this process will become more complex and unclear the larger the model gets. Therefore, a copy and paste function for the Herbal elements would be a useful add-on.

A further suggestion relates to the previously described Soar behavior when applying an already existing value to the working memory in one decision cycle. Soar will remove this memory element from the working memory. If Herbal users with no Soar experience are not aware of this fact, their Soar implementation might crash while running. On the other hand, a Jess program would work without problems. The issue of resetting a memory element to the same value should be addressed in the Herbal tutorial, in the Herbal compiler, or somewhere an interested Herbal user could find it.

The next suggestion refers to the Soar learning functionality. A Herbal user should get a detailed introduction to the Soar learning mechanism in case he intends to develop a learning model. There are rules every Herbal user has to obey when modeling, e.g., the external environment has to be accessed through the top problem space in order to avoid learning two contradictory things. The Herbal programmer should consult the Soar manual first because there are well known problems with the Soar learning algorithm, e.g., recalling desirability preferences, such as best and worst, are not included in the traces of problem solving and therefore have no influence on Soar learning. This can be an advantage, e.g., they are used to create declarative learning effects because they are not learned.

The last suggestion derives from an adaptation that was described in section 3.1.3. The adaptation of the knowledge base was necessary because Herbal does not support nested types. That means that creating an array or a class type is not possible in Herbal. As Soar supports nested types, this describes a gap between Herbal and Soar. This gap can be bypassed by realizing the nested types in Java and using action messages to enter the knowledge. This requires that the Herbal user has access to the environment source code and also has the ability to change it. For example, writing Herbal agents for the Tank Soar environment (a standard Soar environment developed by the Soar community) is not possible because the Tank Soar agent uses nested types to communicate with the environment. When implementing nested types into Herbal it has to be considered that they are not cognitive plausible, and therefore should cost model cycles if their implementation is based on operators.

In addition to the suggestions for Herbal some small bugs were found during the reimplementation of the existing Diag model. The first bug always occurs when using the NOT functionality within a condition for more than one element. For example, if the condition is WORKING-MEMORY-ELEMENT not VALUE1 and not VALUE2, Herbal creates a model that the Soar kernel cannot compile. Another bug comes up while using the graphical user interface. For example after editing a condition within the condition view the interface jumps to a different condition than the one the user just edited. This is inefficient because usually there is more than one change to a condition and therefore the user has to start over searching for this condition. This takes a lot of time in particular with a rising number of conditions. The same effect was also noticed in every other editing view in the Herbal user interface.

3.2. Comparison of the Soar 6 with Herbal

There are different approaches for comparing cognitive models. The first one treats cognitive models like ordinary software and compares for example memory allocation and processor allocation. This leads to a number of indicators that help comparing the models. However, these indicators are not significant for the comparison because they are of little importance for a cognitive modeler. Therefore it is necessary to choose criteria that are directed at the use of cognitive models. The comparison between the existing Diag model and the additional Diag model in Herbal is based on the Diag paper result in combination with the description of Diag model learning capabilities (Ritter & Bibby, 2008).

3.2.1. Adaptations and their influences

Before comparing the existing and the additional Diag model it is necessary to analyze the changes made between the models to explain possible different results. A list of structural changes to the model was provided in section 3.1.3. As mentioned before, these adaptations were necessary for the implementation of the existing model in Herbal. This section discusses why these adaptations might have affected the Diag model behavior and its problem solving predictions. The predictions of the existing model were calculated depending on the number of model cycles that were needed to solve the problem. If an adaptation has an influence on the number of model cycles, it is most likely to influence the Diag model predictions as well. However, it is difficult to examine the influence of individual adaptations in Herbal because the additional model only runs and produces output when all adaptations were implemented. Therefore it would be necessary to find a different way to check the influence of every adaptation on the model predictions.

The first adaptation was the replacement of selected operators with more detailed ones. This leads to a different set of operators that is applied while the model is running. However, the operators are not altering the model's behavior. For example, in the existing model the DECIDE-STATUS operator fires and sets up the conditions for CHECK-LIT. After attending to the component's light the additional model (step 13 to 14 in Figure 10) selects CHECK-LIT-FALSE and then the next operator. The difference between the models is that the existing functionality of DECIDE-STATUS is divided into smaller additional operators. Therefore this adaptation does not lead to a change in how the Diag model predicts the performance while solving the find-fault problem. This can also be seen when looking at the different model traces in Table 3.

The second adaptation was that the transition between problem spaces is no longer managed directly by the Diag operators. Herbal uses transition operators to switch between problem spaces. This adaptation influences the Diag model predicting ability. The number of Soar operators is increased indirectly and therefore more model cycles are needed to solve a problem. This also influences the model's learning ability. An example is that the existing Soar model creates an impasse by firing the INTERFACE-SUGGESTION operator. The additional model needs to learn the INTERFACE-SUGGESTION operator and the problem space transition operator for the INTERFACE-SELECTION problem space. These additional operators also lead to more model cycles and a different learning behavior and therefore influence the model behavior.

The third adaptation was the replacement of internal model knowledge by an external Java based knowledge representation. When comparing the existing and the additional model traces, this adaptation does not seem to have any influence on the model's behavior. In both cases the model reaches the operator that extracts the knowledge, fires it and changes into the next problem space (Table 3). In the existing model DIAGRAM-CHOICE was used to change the working memory by adding the component that had to be checked next. In the additional model CHOOSE-DIAGRAM requests the knowledge and forces the Java environment to add the diagram knowledge to the input link, which is a part of the working memory. Therefore this adaptation should not influence the Diag model behavior.

Table 3. Comparing Soar 6 to Herbal traces (Soar 8).

existing model behavior (Soar 6 trace)	additional model behavior (Soar 8 trace)
1: P: P1 (diagnose)	1: O:O1 (|initialize-Diag-problemspaces-diagnose|)
2: S: S1	2: O: O2 (|Diag-operators-D_Solve-Problem|)
3: O: O2 (solve-problem)	3: O: O3 (|impasse*Diag-problemspaces-find-Faultps|)
4: ==>G: G2 (operator no-change)	4: ==>S: S2 (operator no-change)
5: P: P2 (find-fault)	5: O: O4 (|initialize-Diag-problemspaces-find-Fault|)
6: S: S6 (initial-state)	6: O: O5 (|Diag-operators-FF_Choose-Component|)
7: O: O3 (choose-component)	7: O: O6 (|impasse*Diag-problemspaces-select-Componentps|)
8: ==>G: G3 (operator no-change)	8: ==>S: S3 (operator no-change)
9: P: P3 (select-component)	9: O: O7 (|initialize-Diag-problemspaces-select-Component|)
10: S: S7	10: O: O9 (|Diag-operators-SC_Diag-Suggestion|)
11: O: O5 (diag-suggestion)	11: O: O10 (|impasse*Diag-problemspaces-diag-Selectionps|)
12: ==>G: G4 (operator no-change)	12: ==>S: S4 (operator no-change)
13: P: P4 (diagram-selection)	13: O: O11 (|initialize-Diag-problemspaces-diag-Selection|)
14: O: O6 (diagram-choice)	14: O: O12 (|Diag-operators-DS_Choose-Component|)
15: O: O4 (interface-suggestion)	15: O: O13 (|Diag-operators-SC_Interface-Suggestion|)

Not only adaptations but also changes to Soar itself have influences on the Diag model's predicting performance. The existing model was written in Soar 6. Herbal compiles the additional Diag model into Soar 8. This means that approximately 15 years of Soar development have passed. Following the cognitive architecture idea, this should not lead to any differences in the model behavior. In order to assure that Soar 8 behaves the same way that Soar 6 did a systematic sensitivity analysis should be done to identify which aspects of Soar are crucial for the simulation of empirical results by the Diag model. In particular, all Diag sub tasks should be implemented in the new version of Soar and re-checked for their results. This kind of programming needs to be done in Soar directly to assure consistency. For this thesis, the assumption was made that changes done to the Soar architecture over the last two versions do not influence the Diag model predicting performance.

Another difference between the existing and the additional model is the number of rules. The existing Diag model has 170 rules whereas the additional model only has 72 rules. This difference can be explained by the work which is done by the Java environment, e.g., searching for the previous component within the diagram knowledge. After having discussed possible influences on the predicting performance of the Diag model the next step is to compare actual predictions of the existing and additional model.

3.2.2. Comparing the existing and additional model predictions

Ritter and Bibby (2008) described three possible ways to predict the problem solving time with the Diag model data. They suggested using the number of interface objects examined, the number of model decision cycles with and without learning. Ritter and Bibby (2008) used a regression analysis to find the best indicator for predicting the problem solving time. They tried several combinations of all three measurements within a regression analysis. The results showed that when using the number model decision cycles with learning, the variability in problem solving time accounted for $r^2 = 0.715$ of observations. When using the interface object or number of model decision cycles without learning, no additional variability was reached. Therefore Ritter and Bibby (2008) decided to use the number of model decision cycles with learning as the basis for behavior predictions. This means, when creating additional data predictions the learning ability of Soar needs to be turned on to create the best behavior predictions. As mentioned in section 3.1.3, the additional model has structural adaptations that might

lead to data which does not correlate with the existing data. Therefore an analysis is needed to compare ways of capturing the processing time and compensate the adaptations to the model structure.

The most promising methods for calculating the processing times of the additional Diag mode are:

(a) Counting all model cycles to solve the problem.

(b) Counting all model cycles with operators applied.

(c) Counting all model cycles without the transition operators.

Method (a) counts every model cycle that is needed to solve the problem. It is also the alternative that was used to determine the existing predictions. However, it does not take into account any of the adaptations that have been made to the additional model. Method (b) is based on the idea that an adaptation, e.g., the use of the external knowledge base, leads to more NO-STATE-CHANGE cycles because the model has to wait for the environment response. Method (c) is based on the idea that the existing model did not have transition operators and therefore these model cycles should not be included.

Linear regression was used to examine which of the three methods for calculating the processing time provides the best match to the existing Diag problem solving prediction: (a), (b) or (c). Given that all three methods are not completely independent; the results can only differ in the additional model learning behavior. The available predictions from the existing model are restricted to the 10 stimuli sets used in Ritter and Bibby (2008). A stimulus is a set of 20 trials that consist of PS as start fault and 19 random faults. The exact order of the 10 stimuli sets can be found in Ritter and Bibby (2008). Using the three methods described above, the additional model had to solve the same 10 stimuli and generate behavior predictions.

The predictions from the existing model and the additional model were compared and the results from the regression analysis are shown in Table 4. Method (b) and (c) predict the variability in the existing model processing time for solving the fault finding task with a correlation of $r = 0.78$. Method (a) reaches $r = 0.8$. The results support two conclusions. First, the difference between (a), (b), and (c) is minimal and therefore the additional model's learning behavior does not have strong influences on the variability of the model behavior. Second, the additional model does not behave in the exact way the existing model does. Both conclusions ask for a detailed comparison between the two models and thus the examination of an example stimulus is necessary.

Table 4. The results of the linear regression of methods (a), (b), and (c) over all stimuli.

Method	R^2	Intercept	B
(a) Decision cycles	0.649	57.205	1.389
(b) Decision cycles with operators	0.617	0.357	0.998
(c) Decision cycles without transition operators	0.618	0.369	1.021

For a detailed comparison, the separate stimulus 2 was analyzed using regression analysis and the results are shown in Table 5. The results are similar to the complete analysis in Table 4 and thus contain the same suggestions. Table 6 provides a list of faults next to the number of decision cycles for both models. Method (a) was used to count the additional model cycles. For a better comparison of the

models' differing behavior the ratio between the two models was calculated and is also shown in Table 6. If the models had the same behavior, the ratio would be a stable value with a fluctuation of less than 10% because of the model's statistical behavior. The ratio has an average value of 3.23 and a standard deviation of 2.36. This is a fluctuation around 65% and thus the models have different learning behaviors. This is especially reflected in the highest ratios (ID 18 to 20). Thus a more detailed analysis of the models' learning behavior is provided in section 3.2.3.

Table 5. The results of the linear regression of methods (a), (b), and (c) for stimulus 2.

Method	R^2	Intercept	B
(a) Decision cycles	0.743	57.139	1.472
(b) Decision cycles with operators	0.703	63.617	0.828
(c) Decision cycles without transition operators	0.704	63.103	0.839

Table 6. The task number (ID), the faulty component (Fault), the models decision cycles and their ratio between them for stimulus 2.

ID	Fault	Existing Model Decision Cycles	Additional Model Decision Cycles	Ratio (Additional /Existing)
1	PS	68	91	1.63
2	EB1	105	137	1.25
3	SA2	275	375	1.50
4	EB1	38	43	1.13
5	LB	198	369	2.13
6	SA1	84	325	3.56
7	EB2	60	183	3.48
8	MA	58	215	4.16
9	SA1	35	114	3.03
10	SA2	45	127	3.00
11	EB2	32	64	2.25
12	EB2	15	64	4.80
13	EB1	13	43	3.31
14	MA	31	77	2.74
15	LB	70	132	2.11
16	PS	31	30	0.97
17	LB	83	132	1.78
18	MA	10	77	8.50
19	SA1	18	114	5.89
20	SA2	15	127	9.00

3.2.3. Speed and learning mechanisms

The comparison between the existing and additional Diag model was based on performance predictions of both models. Ritter and Bibby (2008) were able to show that the best Diag performance predictions were based on the Soar learning mechanism. As described in section 2.3.1, the Soar learning mechanism is based on chunking. This learning mechanism is most effective when a task is done several times with the same sub problem spaces. While the Diag model is running, it

continuously changes the model state to sub problem spaces and returns results to higher-level problem spaces. This leads to the creation of new chunks that are added to the production memory. Those chunks are matched in similar situations to avoid calling the sub problem space. That way, the sub goal state is reached right away.

The comparison in section 3.2.2 came to the conclusion that the existing and additional model have different learning behaviors. Thus a closer look at the learning mechanisms of both models is needed. Soar's learning mechanism creates three types of chunks while running the existing Diag model. "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." (Ritter & Bibby, 2008, p. 11). These three types can be associated with:

- Procedural knowledge → specific knowledge about how to apply an operator through search in a sub problem space
- Episodic knowledge → creation of an operator for use in a higher problem space
- Declarative learning → creation of state argumentation rules which add derivable knowledge to a state

The existing model consists of 173 rules (172 model rules and 1 problem state rule) that are loaded in the Soar kernel when the model is started. The number of chunks created while solving one stimulus set depends on the order of trials. For example, if the broken component while doing the task 20 times is always PS, the model would presumably stop learning after four trials. While performing a random order of 20 trials the existing model learns around 200 out of the 287 rules that it can learn as a maximum. The existing model needed around 24 random tasks to learn all possible rules.

The additional Diag model also uses the Soar learning mechanism to learn chunks and increase efficiency while solving the fault-finding task 20 times. The additional model consists of 79 rules that are loaded in the Soar kernel when the model is started. As in the existing model, the number of chunks created while solving one stimulus does depend on the order of trials. However, when the broken component while doing the task 20 times is always PS, the additional model stops learning after 2 trials. While performing a random order of 20 trials the additional model learns 186 chunks. This is also the maximum number of chunks it is able to learn. The number of trials needed to learn all chunks is approximately 9 and does not depend on the order of the fault-finding problems.

The acquisition of knowledge in the additional model is less distinctive than in the existing model. This can be explained for procedural knowledge. The existing model elaborates the INTERFACE-CHOICE operator in a way that it suggests checking PS when the previously checked component is NONE. This leads to a procedural operator implementation. With the additional model's INTERFACE-SUGGESTION operator this kind of learning never occurs because this operator sets up the INTERFACE-SELECTION problem space and thus never results in a procedural operator while learning. The transition operator from SELECT-COMPONENT to INTERFACE-SUGGESTION, which should be an equivalent to the existing model's INTERFACE-SUGGESTION operators, produces no such knowledge.

By looking at the ratio between the models performances in Table 6 it appears that the ratio is rising when the additional model stops learning while the existing model is still learning. Table 7 is comparing the highest ratios of every stimulus. The highest ratio depends on the faulty component and its place within the 20 trials. The ID for the highest ratio within a stimulus always lies between 15 and

20. The faulty component is either MA (4 times) or SA2 (6 times). This data supports the assumption that the additional model stops learning earlier in a stimulus than the existing model.

Table 7. The stimulus number, the task number (ID), the faulty component (Fault), the models decision cycles and their highest ratios in all stimuli.

Stimulus	ID	Fault	Existing Model Decision Cycles	Additional Model Decision Cycles	Ratio (Additional /Existing)
1	15	sa2	15	135	9.00
2	20	sa2	15	135	9.00
3	16	sa2	15	135	9.00
4	18	ma	10	77	7.70
5	15	ma	10	85	8.50
6	15	sa2	15	127	8.47
7	17	ma	10	85	8.50
8	20	ma	10	77	7.70
9	19	sa2	15	127	8.47
10	18	sa2	15	135	9.00

3.2.4. Summary of the comparison

The comparison between the existing and additional model brought interesting insights on how the models are predicting performance, why they create different results, and how this might affect the quality of the predictions. These three questions are important for the further research on the Diag model and within this thesis especially for the research on strategies and strategy shifts.

The existing model predicts problem solving times with a variability of 79% compared to human performance. Since the regression was not significant for 20% of the participants (Ritter & Bibby, 2008), this variability reaches 95% when these 20% are removed from the existing data. Three ways of calculating the additional model's processing times were discussed and the number of decision cycles was chosen as the best alternative. However, the additional model did only correlate with $r = 0.8$ to the existing model's behavior and is therefore seen as a different model. This also leads to the conclusion that the additional model creates different predictions for human behavior.

In order to analyze why the existing and the additional model produce different predictions the ratio of their decision cycles was calculated. By analyzing the highest ratios and mean ratios it showed that the difference is based on the models' learning abilities. While solving a stimulus the existing model learns up to the last quarter (trial 15 to 20) and even continues after the twentieth trial. The additional model stops learning after 9 trials independently from the order of problems because it does not use the full range of learning types. This suggests that the additional model has problems with the Soar bottom-up learning mechanism. This could result from an incorrect implementation or a bug in Herbal itself. The problem could also lie in the current version of Soar.

Before using the additional Diag model for further research a comparison between its predictions and human data is necessary to identify how significant the model is. This comparison is provided in section 3.3. Besides the significance analysis, the additional model results cannot be used to make definite statements about the correctness of the Diag model. This means the result of this thesis have to be analyzed carefully for their use to support or refuse the Diag model.

3.3. Analyzing existing user data with the additional model

Since both models produce similar, but different results, it is necessary to analyze the existing user data and the additional model predictions. Therefore this analysis examines how significant the predictions of the additional model are. The following questions were chosen to examine how significant the additional model is:

1. How accurate is the model in predicting specific examples of tasks?
2. How accurate is the model in predicting participant performance per trial?
3. How accurate is the model in predicting the general learning profile?

Answering these questions is also necessary for the evaluation of the additional model concerning its usability for further research. Section 3.2.2 showed that the best way to compare the existing and additional model is by using the number of decision cycles. Therefore this analysis is based on the additional model cycles with learning. By using linear regression between the additional model predictions and the existing user data an average reaction time (2.4 s) and an average time as slope of decision cycles (0.062 ms) was calculated. For each of the following analyses the predicted times (as slope of decision cycles * decision cycles + intercept = 0.063 ms * decision cycles + 2.283s) are compared to the observed problem solving times.

In the diagrams that contain mean values an additional value was determined and presented by using error bars. The error bars in this thesis show the standard error of the mean. The standard error is a statistical dispersion for a sample distribution. Equation 1 shows how the standard error for a sample of size N is defined. Since the standard deviation is thereby adjusted to the sample the error bars are more related to the test of significant differences than the standard deviation alone.

$$SE_{\bar{x}} = \frac{S}{\sqrt{N}}$$

where

S = the standard deviation of the sample

N = the size of the sample

Equation 1. Standard error of the mean.

Figure 11 shows the comparison between the predicted and observed average times and standard errors for each fault aggregated over the existing user data and the additional model predictions. As in Ritter and Bibby (2008) there is a general increase for the predicted and observed times in time taken to solve the problem. These results depend on the strategy that most people use to solve the problem when working through the interface from left to right. The correlation between the existing user data and the additional model prediction is r = 0.79. Figure 11 shows that the more on the right side of the interface the fault is positioned, the more the predicted and observed times increase.

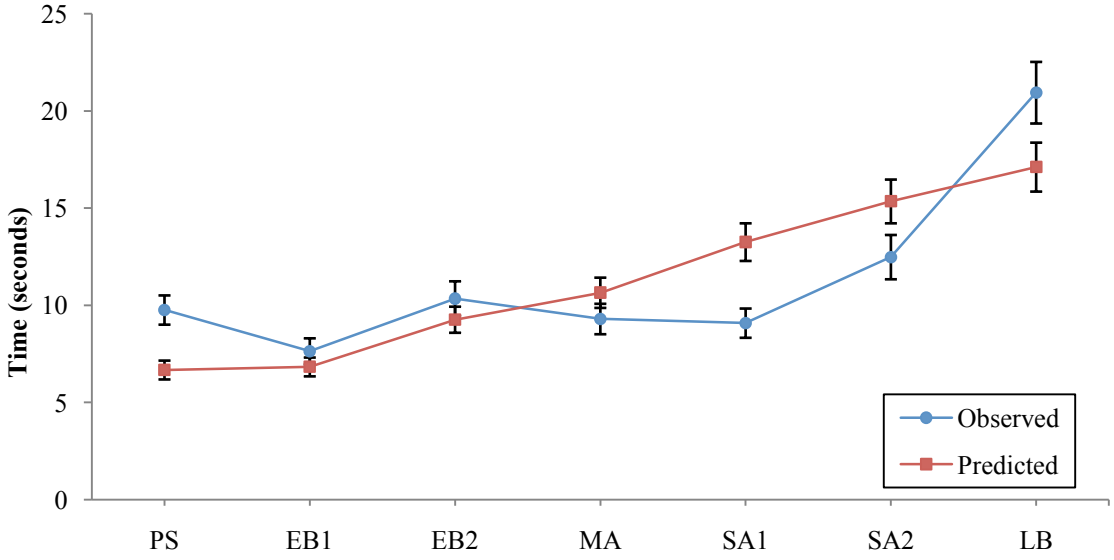


Figure 11. The observed and predicted problem solving times (means and standard errors) for the seven different faults averaged over participant and trials.

Figure 12 shows the times the model predicted and the observed times over the series of 20 trials. All trials are aggregated over participants. Ritter and Bibby (2008) pointed out that the curve is not a monotonically decreasing learning curve because the first two faults are not completely random in the order of a stimulus. Beginning with the third component the faults are completely random. This leads to a monotonic increasing curve within the first 3 trials. After that the normally decreasing learning curve is visible. Overall, the additional model's predictions across trials correlate with r = 0.89.

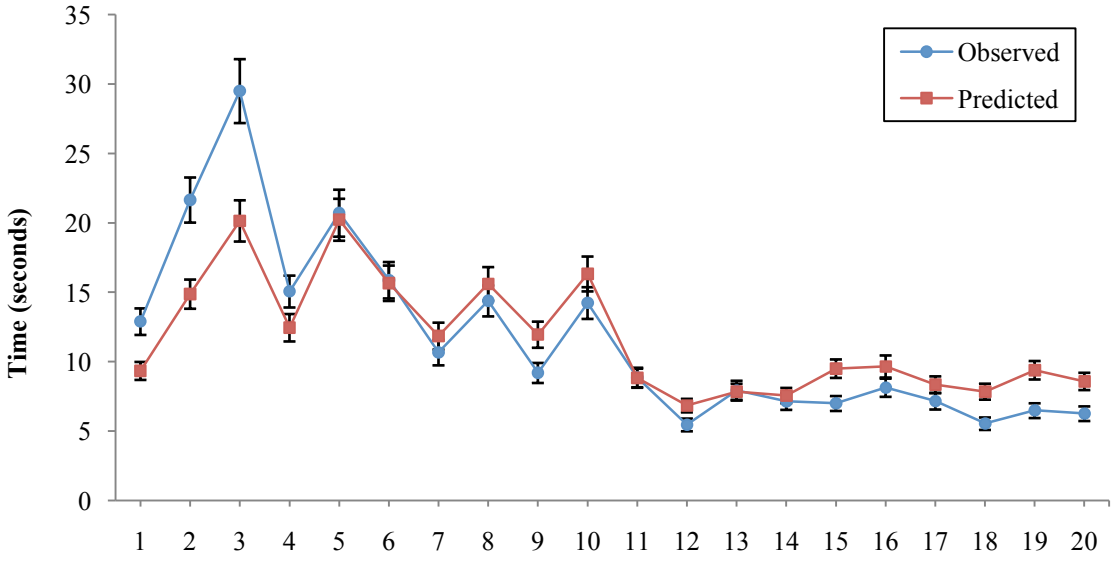


Figure 12. The observed and predicted problem-solving times (means and standard errors) for the 20 trials averaged over participant and fault.

Table 8 shows that there also is a relation between the participant performance and the predictions over every stimulus. As mentioned before, a stimulus is a random order of 20 trials with different faulty components. Each participant saw a different order of 20 trials.

Table 8. The correlation (R), intercept (in seconds), regression coefficient (B, in seconds per model cycle), and number of correct responses for each participant when cycles are regressed onto problem solving time.

Participant	R	Intercept	B	N
P1	0.781	1.759	0.058	19
P2	0.868	-0.589	0.095	16
P3	0.738	2.415	0.078	19
P4	0.836	1.972	0.065	20
P5	0.519	3.488	0.007	20
P6	0.829	1.418	0.083	18
P7	0.261	10.221	0.015	19
P8	0.831	1.708	0.070	19
P9	0.861	-0.559	0.089	17
P10	0.808	2.184	0.065	18
Mean	0.733	2.4	0.063	18.5

The comparison of the additional model and the existing data showed that the additional model was able to predict the participant performance to a certain extend. The predictions from the additional model are not as accurate as the ones provided by the existing model but they do have similar behaviors. These similar behaviors can be seen at the fact that both models managed to predict human behavior well. However, when comparing the correlations for the predictions per fault, per trial, and per participant the additional model is constantly 10 to 15% less accurate than the existing model. Another similarity is that both models have problems in predicting the performance of participant 5 and 7. Ritter and Bibby (2008) suggested that participant 5 was solving the stimulus with less learning. Since the additional model is also performing the task with less learning, its prediction about the individual performance of participant 5 is about 0.25 more correlated then the prediction of the existing model. Therefore the development of several strategies is necessary for a detailed analysis of the performance of these two participants.

Differences between the existing and the additional model predictions are almost entirely caused by the different learning behaviors. For example, Figure 13 shows that the existing and additional predictions for the existing data continually grow apart after trial 11.

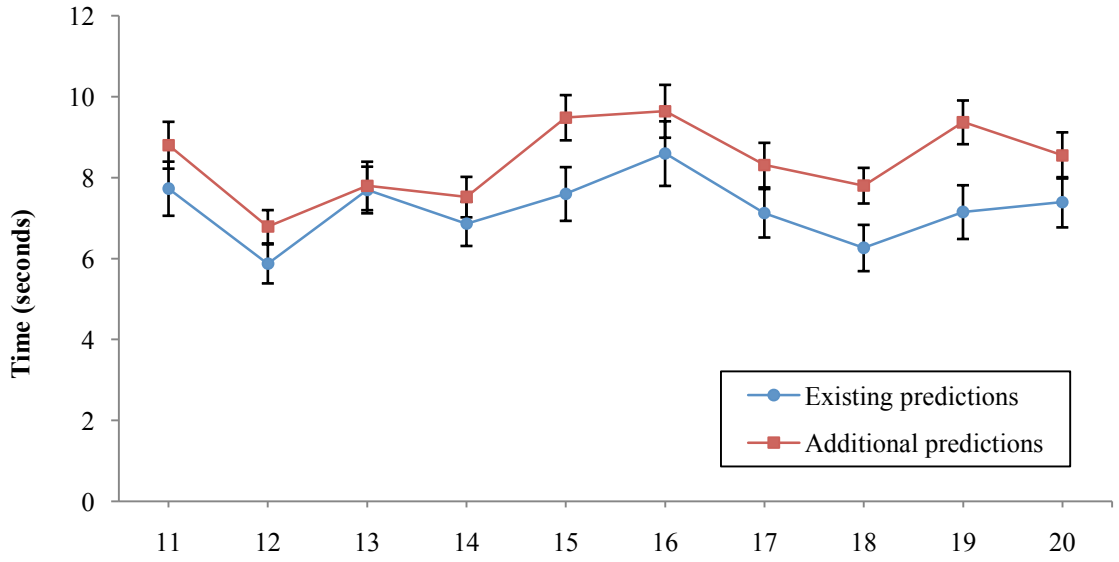


Figure 13. Predicted times (means and standard error) for the trials 11 to 20 averaged over additional and existing model.

3.4. Summary

The different types of additional Herbal models described in section 3.1.1 were implemented within 8 weeks. This is much shorter than the time needed to develop the existing model (6 months according to the source code). Since there is a distinction between developing and implementing a model, taking the speedup that was achieved with Herbal is not accurate enough to be significant. Herbal's strength is that it makes changes to the additional model easy and therefore different types of the additional model could be developed relatively quickly. The reuse of problem spaces, operators, conditions, and actions was helpful and will be helpful for the development of different strategies. A possible improvement for Herbal could be a tutorial, guideline, or design patterns for helping developers implementing models that are as complex as Diag. Without such help, it was difficult to find the correct method of implementation and led to problems especially with Soar's learning mechanism.

The implementation of the additional Diag model is a major part of this thesis. Overall, the additional Herbal model appears to implement the same strategy that the existing model and most of the participants used. By following the Ritter and Bibby (2008) approach of predicting problem solving times by examining the decision cycles needed to solve a problem, it was found that the additional model fit the existing data when examining specific fault tasks. The best correlation between the additional Herbal model and the existing model was found when comparing the number of decision cycles with the Soar learning mechanism turned on. Assuming that the existing model produced good predictions, additional predictions for the existing user data were calculated by using the additional Herbal model decision cycles with Soar learning turned on. This additional data turned out to fit the user performance not quite as well as the existing predictions, but still well enough to model human behavior. These results allow proceeding with the research on strategies and strategy shifts (chapter 5).

4. GATHERING ADDITIONAL DATA

For the research on strategy shifts it is necessary to gather enough data that facilitates a significant result. Ritter and Bibby (2008) published 10 participants' reaction times while solving the fault-finding task. For two participants the Diag model failed to predict the reaction times and it was suggested that these participants used a different strategy than the Diag model. This suggestion is also supported by the fact that the additional Diag model with its different learning ability was able to predict participant 5 with an around 20% higher correlation. The data set of these two participants needs to be extended with additional data. In order to find more participants like participants 5 and 7 from the existing study data, additional user studies with the diagrammatic reasoning task were run.

4.1. Experimental Methodology

4.1.1. The participants

The experiment was performed with the help of 37 participants that differed in sex (7 female and 30 male), age (between 18 and 29 years), and nationality (16 Americans and 21 Germans). The participants were separated into 11 stimuli groups. Every group was given a random order of 20 trials for the fault-finding task. Each group consisted of 3 participants, except group 11, which consisted of 7 participants. All participants were undergraduate students from the Pennsylvania State University or the Otto Friedrich University in Bamberg. Their majors were psychology (8), computer science (12), economics (5), business informatics (8), and mechanical engineering (4) (Appendix 5 – Participants and their majors). The Americans were paid $7 for participating. All participants had a basic understanding of physic and electric circuits which made the introduction to the fault-finding task relatively easy, but none of them was familiar with the actual task. Participant IDs that will be used later in this thesis were given according to the following principle: D14_10 stands for the 14th participant in Germany who solved stimulus 10 while S1_3 stands for the first participant in America who solved stimulus 3.

Each participant attended one experimental session that lasted approximately 30 minutes, depending on the participant performance. All participants were instructed that they were free to withdraw from the research program at any time without penalty. If they chose at any time not to participate further, they would be paid for the amount of time of actual participation. No participant quit before completing the task. All the data gathered as part of this thesis was anonymized before analysis.

4.1.2. Materials and Apparatus

The materials used for the study can be separated into introduction forms, task environment introduction, and task environment. The introduction (Appendix 1 – Written instructions for the participants) consisted of 3 parts and intended to be as close as possible to the user study run by Ritter and Bibby (2008). The first part tries to create a background story to motivate the participants and make them susceptive for the fault-finding task. The second part consists of a schematic of the underlying circuit, a picture of the control panel with all switches off and no light on, and a picture of the control panel with all lights on. In the existing user study an example picture with PS as a faulty component was provided as well. This time, in the intention to create different strategies and strategy shifts, no participant should get an example of the task. The third part of the instruction is a set of

frequently asked questions and their answers. These questions were collected from pilot participants and undergrad students from a human computer interface class at Pennsylvania State University.

The task environment introduction (Appendix 2 – Task environment introduction) and the task environment were PDF files presented with the Adobe Acrobat Reader 8 on a MacBook Pro. Both files are exactly the same except that the task environment introduction has no fault-finding task information (interface). The environment introduction was used to prepare the participants for the actual study environment and to minimize their adaption time to the environment. The task environment consists of 43 slides (2 preparation slides + 20 task slides + 20 pause slides between the task slides + 1 Thank you slide). Appendix 4 – Task environment example shows an example task slide. Participants had to use the Space or Arrow Down key to switch to the next slide. For navigation on a task slide the participants had to use a one button MacMouse.

The participants' performance while solving a stimulus was captured with a keystrokes capture software called RUI. This open source software for Mac and Windows allows a capturing accuracy up to nanoseconds and is available for free at Ritter (2008). The software creates an ASCII file with all mouse and keyboard actions and their associated time stamps within the captured time. An example of the RUI output stream is shown in Table 9. RUI captures mouse movement by saving every new mouse position. This means the output file is ordered by the elapsed time. A separate java application called DataAnalyser was developed to analyze the RUI output files and filter their contents. This software steps through the user's actions documented in the ASCII file and looks for a specific pattern. The DataAnalyser collects the participant times for every trial, the answers given, the mouse movement on every trial sheet, and the number of mouse movements per trial sheet. The given answers were compared to the correct answers. The result of this comparison and all the data collected was stored in a Microsoft Excel file for further analysis. This means that besides the collected information, every Excel file contains the information whether the participant was correct or not.

Table 9. Example of RUI data capturing.

Elapsed Time (nanoseconds)	Action	X (screen position)	Y (screen position)
0	Moved	737	63
972171	Moved	736	64
1039183	Moved	731	74
1072759	Moved	727	74
9077609	KEY	COMMAND	SPACE
41450617	Pressed	Left	
41578422	Released		Left

For the user study at the Pennsylvania State University an eye tracking device was used to capture additional user data. The eye tracker produced a video output which shows a view through the left eye and a target cross shows what the participant is looking at. Figure 14 shows frames from the video stream which were captured using a VCR connected to the head mounted camera. The information from the eye tracker was analyzed manually by watching the videos. Attention was put on the order in which the participants attended to the interface information and how fast they managed to navigate through the trials.

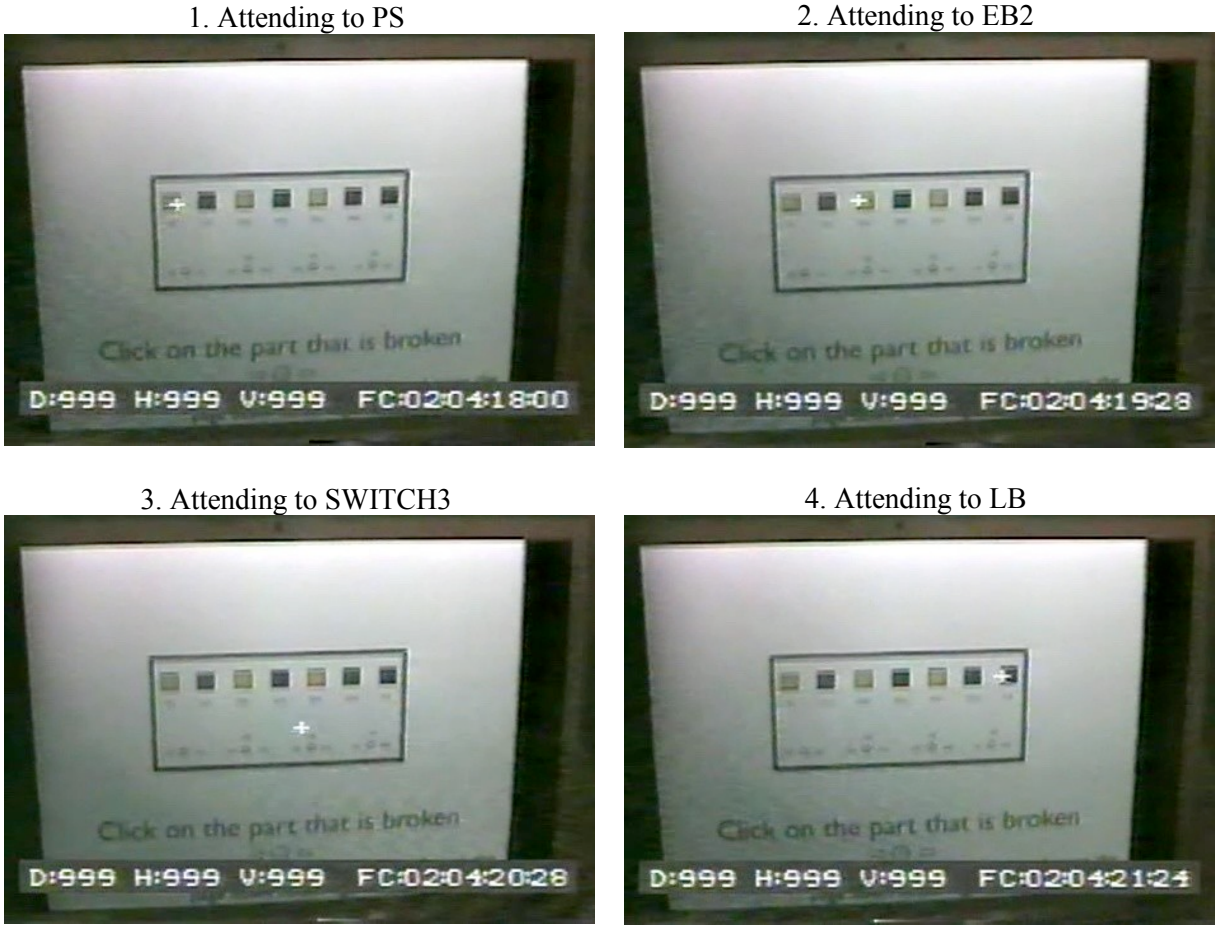


Figure 14. Participant S13_3 solving fault-finding problem with LB as the faulty component.

4.1.3. Procedure

The procedure of the study was based on the Ritter and Bibby (2008) procedure. At the beginning every participant was asked to read the instruction material. While studying the instructions they were allowed to ask questions about the experiment. The participants were told to especially pay attention to the schematic representation of the electrical circuit with its switches and components (Appendix 1 – Written instructions for the participants, p. 2). After a study period of 5 minutes the participants were asked to draw the schematic without looking at the template. If the participant was unable to reproduce the schematic an additional 5 minute study period and a second drawing trial was given. No participant needed more than 10 minutes to accurately reproduce the schematic and about 80% of the participant succeeded in the first trial. After the reproduction of the schematic the instructions were removed and the training phase began. If the participant agreed to use the eye tracker system, the adjustment of the eye tracker was done before starting the training phase.

The training phase was performed on a MacBook Pro. The participants were asked to put their regular mouse hand on the MacMouse and their other hand on a slide down key (either SPACE or PAGE-DOWN). The participants had to work through 5 task trials (Appendix 2 – Task environment introduction) and pretend to click on the faulty component in order to train them in clicking and changing the slide. This was necessary to minimize possible incorrect behavior and to maximize their mouse clicking and slide changing performance while running the actual study. It should also give them the ability to ignore the environment and concentrate completely on the fault-finding task.

Before starting the task, the participants were told to keep the following things in mind:

1. Within the 20 trials, a component could be faulty more than once.
2. No direct response to an answer is given. However, if more than 3 mistakes within the first 5 trials occur, the experimenter will stop the experiment and provide a strategy.
3. If possible, the participants should follow their thoughts with the mouse(Ritter & Bibby, 2008).
4. No double checking should be performed. This means after figuring out which component is faulty the participant should click right away.

After the environment introduction the task environment was loaded and the RUI software began capturing the mouse and key actions. After the introduction slide there was a mouse conditioning slide (Appendix 3 – Mouse conditioning). The participants were asked to click on each number or the corresponding circle. This task served as MacMouse training and as a marker in the captured mouse and key action stream produced by the RUI software. After the mouse conditioning slide the first task slide was presented. When the participants had solved all 20 trials the RUI software was stopped.

After the participants had finished the fault-finding task they were asked about their major, their MacMouse experience, and their thoughts about the task. Also, the participants were shown which of their answers were correct. The participants were debriefed and thanked.

4.2. Results

The amount of collected data allowed the analysis of the mouse movement, the eye movement, and the reaction times to study the participant behavior while solving the task. To support chapter 5, the major goal of the analysis is to find participants with strategies other than the Diag strategy. The user study results are based on the analysis of 35 data sets because two participants had an error rate above 25%.

A participant data set contains of mouse movement data, the eye tracking video, and the reaction times. The mouse movement and reactions times were gathered by using the RUI software. The eye tracking video was captured using a VCR. When filtering the RUI action stream the DataAnalyser created a Microsoft Excel file with two separate table sheets for every participant. The first sheet contains the mouse movement for each of the 20 trials, the number of mouse position changes, and the time from the first position change in a new trial to the click on a component (section 4.2.1). The second table contains the reaction times and answers (section 4.2.3). For the analysis of the eye tracking video the tape was digitalized into several video files (section 4.2.2).

4.2.1. Mouse movement

Mouse movement is specified by the change of mouse position on the x and y direction on the computer monitor. Every mouse pointer change is captured and the x and y coordinates are saved together with a time stamp (Table 9). Within one trial, the mouse movement begins when the mouse is first moved and ends with the click on a component. A reference point on every task slide was used to orientate the mouse movement at a starting point. The participants were told to return the mouse to this point before changing to the next task slide.

In order to increase the significance of the mouse movement data the participants were asked to follow their thoughts with the mouse. Unfortunately, not all participants did use the mouse this way. Therefore, the participants that used the mouse movement to support their thoughts have to be

separated. To do so, a clustering algorithm was used that takes the individual mouse movement as indicators, and separates the participants into groups. The average number of mouse movements, the average time for mouse movement, and the standard deviation of both values were calculated for every participant. These indicators served as feature vectors for the k-means clustering algorithm. This algorithm is used to cluster N objects based on attributes into K groups with K < N. It assumes that the object attributes form a vector space. It tries to minimize total intra-cluster variance. K-means was initialized with three random initial vectors because three types of mouse movements were identified when analyzing the data manually. The results of the k-means algorithm and the resulting groups with the respective participants are shown in Table 10.

Table 10. K-means results used to cluster the mouse movement behavior into groups.

Group	(a)	(b)	(c)
Number of participants	17	13	5
Average distance to the centroid	14.1	22.1	34.3
Participants	D1_7 S13_3 D2_8 D12_8 D19_11 D7_3 D8_4 D17_11 D16_11 S9_7 D6_2 D18_11 D4_10 S2_9 S10_6 D9_5 D14_10	D3_9 S14_4 S3_8 S8_10 D11_7 D21_11 S16_6 S11_1 D10_6 D15_11 D13_9 S15_5 S12_2	S1_3 S6_2 D20_11 S5_4 S7_5

The groups shown in Table 10 can be described as follows:

(a) Participants who used the mouse for clicking directly on the faulty part. Their movement routes can be described as a straight line from the reference point to the component.

(b) Participants who used the mouse to support the problem solving. Their movement routes can be described as their path of thoughts while solving the task.

(c) Participants who used the mouse with no clear intentions. Their movement routes do not follow a clear order.

Figure 15 shows example movement routes which support the results from the k-means algorithm.

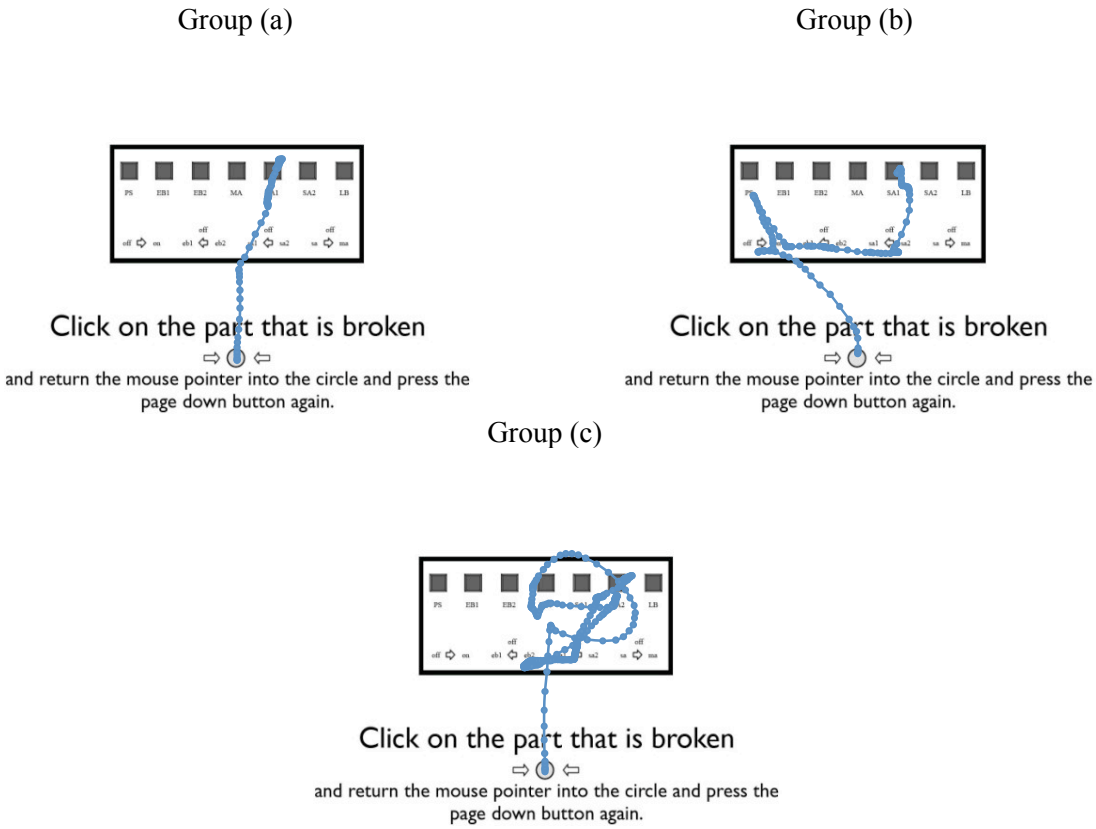


Figure 15. Example trials from every mouse movement group, solving fault SA1 in trial 3 of stimulus 11.

Due to the effects of learning all participants changed their mouse movement behavior while solving the stimulus, e. g. participants from group (a) moved even more directly than in their first trials, or some participants from group (b) switched to group (a) within their last trials. However, the group the participants were assigned to represents their prevailing mouse moment behavior. Therefore only the participants in group (b) could be used to define additional strategies because group (a) made the identification of a strategy impossible. Group (c) could not be used even if participants switched to group (b) during their last trials because they had optimized their original strategy too much. The participants from group (b) allowed the identification of three strategies.

The first strategy was the same as used by both Diag models. Figure 16 shows examples of such behavior. The participants tended to first check the lights and then, if necessary, the switches and the previous component.

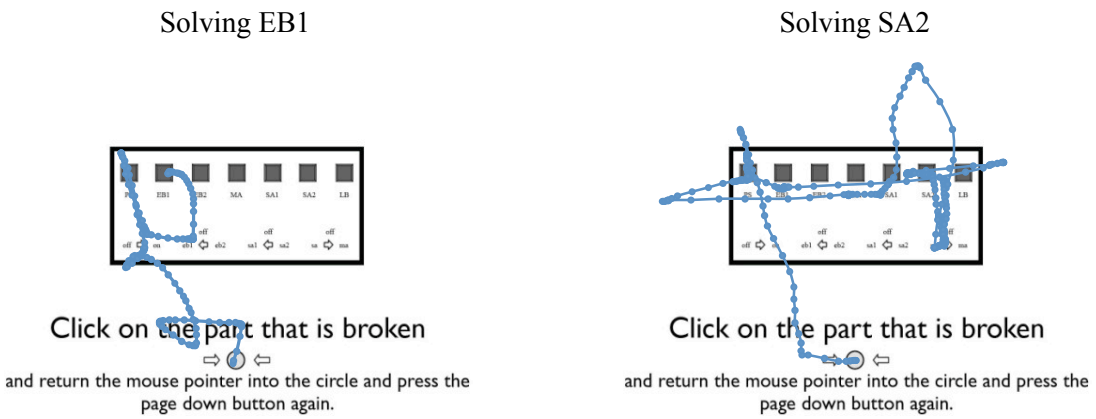


Figure 16. The mouse movement of participants that solve the fault-finding task with the Diag strategy.

The second strategy is named PrevState because it is based on the transition from one trial to the next. This strategy uses two directions when attending to the next component. Participants used the previous fault to begin the problem solving process and then moved into the direction of the fault. Examples of both directions are shown in Figure 17. If, for example, the previous fault was EB1 and the current fault was behind EB1 in the circuit, the participants started by checking MA or SA1. It is unclear which kind of testing algorithm was used to determine the direction or how the component in the middle of the circuit was tested.

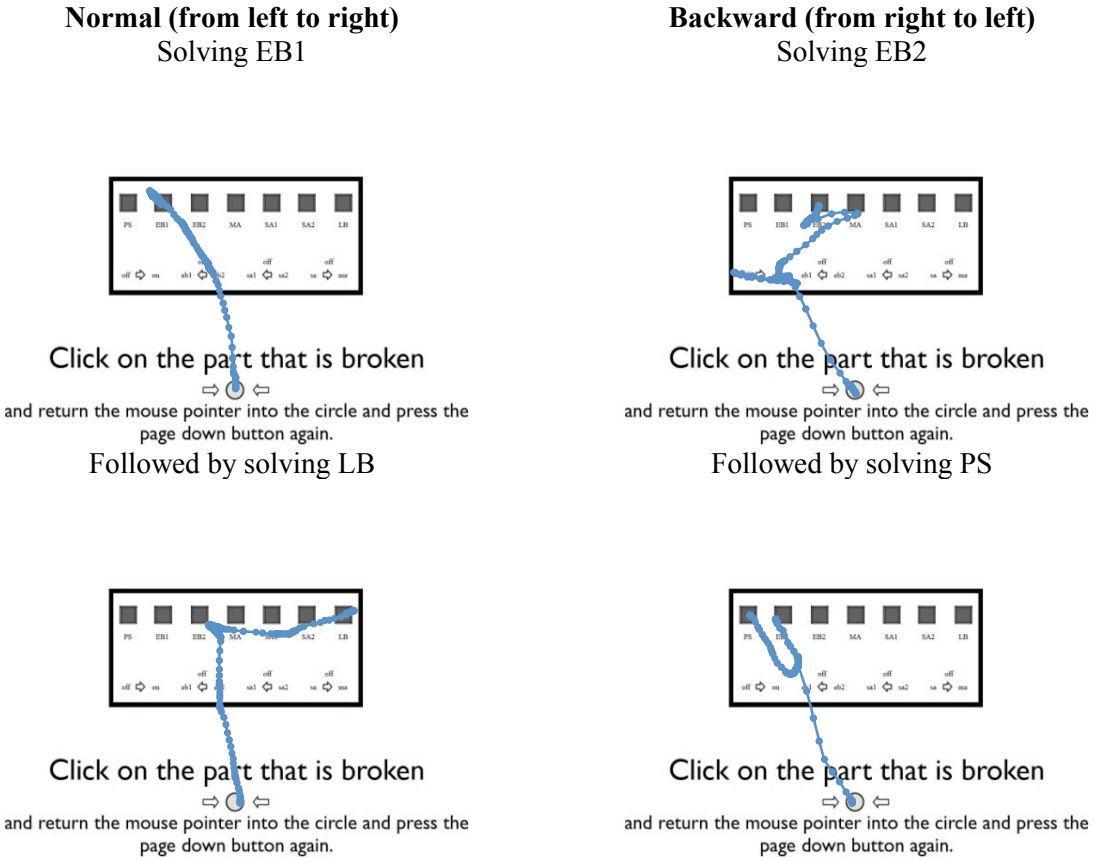


Figure 17. Mouse Movement of different participants using the PrevState strategy to solve the fault-finding task.

The third strategy is the DiagSelect strategy. It solves the fault-finding task by moving through the circuit beginning with PS and using the switches as orientations. By orientating at the previously checked component the participants were attending to the next switch in the circuit. Depending on the switch position the next light was checked. When the faulty component was reached they clicked on it. An example for this behavior is shown in Figure 18.

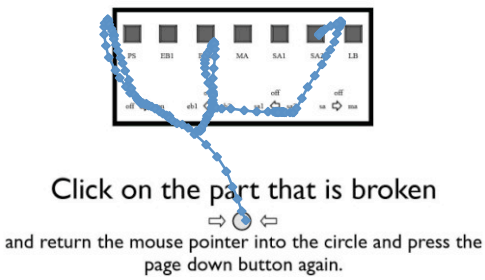


Figure 18. Participant solving the fault-finding task with the DiagSelect strategy and SA2 as fault.

It was discovered that all three participant groups performed less mouse movement in the last quarter of the stimulus. Anderson's (1995) stages of skill acquisition can be seen in all mouse movement sets, especially in group (c). Figure 19 shows the average number of mouse movements per trial with the according trend function with a negative slope. This suggests that even after the mouse training task (Appendix 3 – Mouse conditioning) the participants still improved their mouse movement performance while solving the task.

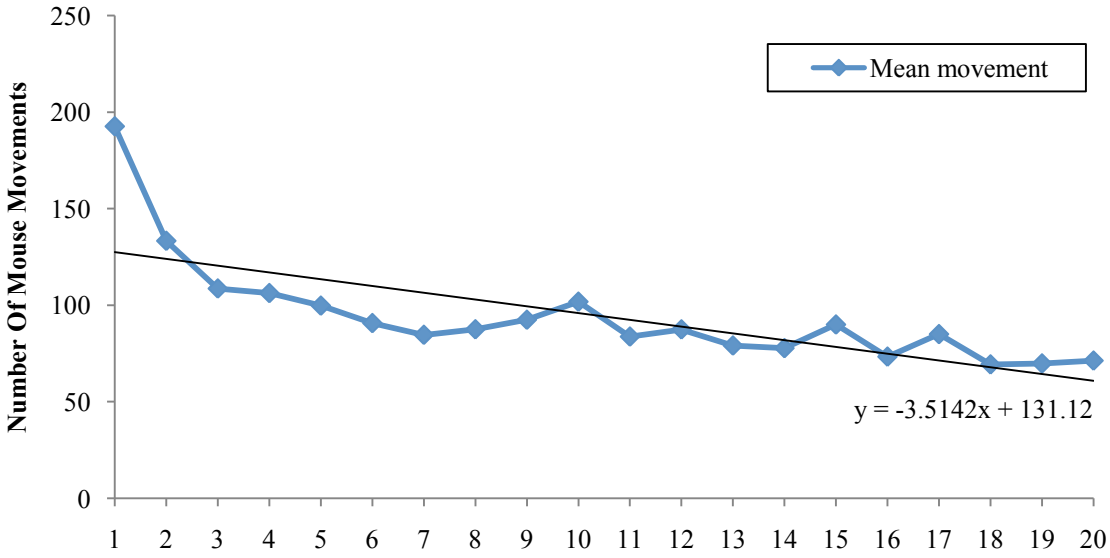


Figure 19. Number of mouse movements per trial and the corresponding linear trend function with a negative slope.

Card et al. (1983) used the mouse movement and clicking time to analyze the improvement on the learning rate. In order to show improved performance in the mouse movement of the additional study it is necessary to calculate the average mouse movements. Card et al. (1983,chapter 2) published the Power Law of Practice to predict this behavior. Equation 2 can be used to predict the performance improvement for the average number of mouse movements per trial or the average time for every trial.

$$\log T_N = \log T_1 - \alpha \times \log N$$

where:

T_N = estimated mouse movements on the N trial
or estimated time per trial N

T_1 = estimated mouse movements on the first trial
or estimated time for the first trial

N = trial number

α = an empirical determined constant

Equation 2. Power Law of Practice (Card et al., 1983).

Therefore the effect of learning by using the mouse to select the faulty components can be described by the values T_1 and α. These values are determined by a linear regression between $\log T_N$ and $\log N$ for mouse movements and trials or movement times and trials. The results of the regressions and the learning functions are shown in Table 11. The learning functions in Table 11 were used to plot the learning rate compared to the actual values in Figure 20. Both values show similar results compared to the learning rate.

Table 11. Result of the linear regression between $\log T_N$ and $\log N$.

Learning	R	Intercept	B	Learning Function
Movement Times	0.92	13.76	0.44	$T_N = 13.76 * N^{-0.44}$
Mouse Movements	0.93	159.90	0.27	$T_N = 159.90 * N^{-0.27}$

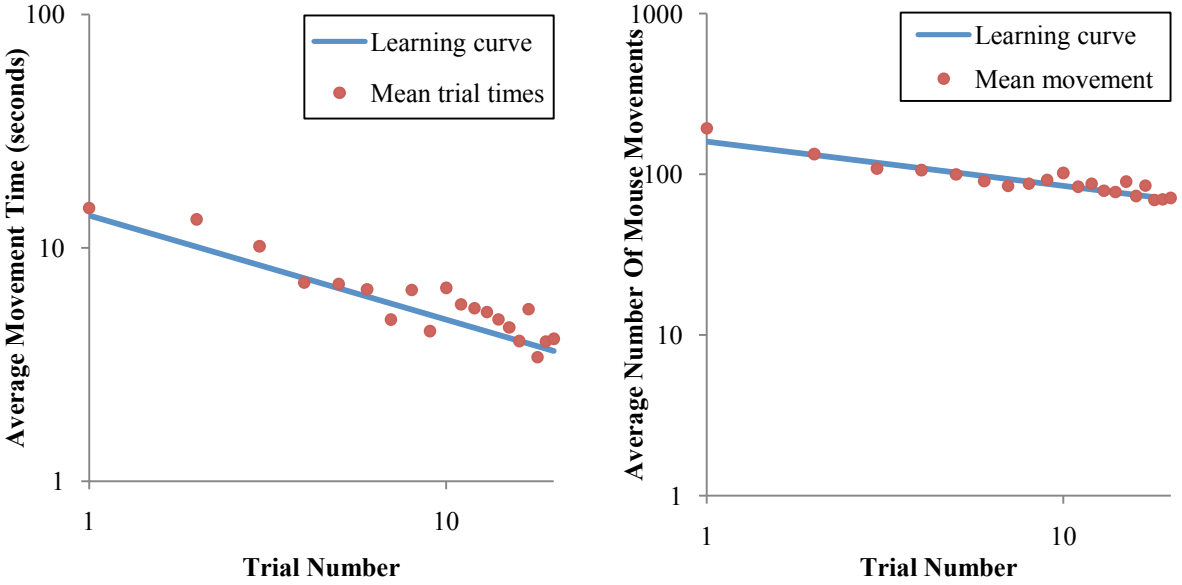


Figure 20. Learning curves and mouse movement over trials.

4.2.2. Eye tracking

Nine undergraduate students from the Pennsylvania State University had their eye movement captured while solving the fault-finding task. Three of the resulting video files could not be used for further analysis because the eye movement was not constantly visible. The significance of the movement data for the cognitive process is hard to define since there is a difference between looking at a component and noticing a component. Another problem is that eye tracking is limited to a quantitative function, e.g., while looking at PS for a longer time participants might think about something else. Also, it is difficult to identify what participants really notice since they might gather information indirectly by using periphery angle of view. For example, when changing to a new slide the participant can approximate the number of lights that are lit up without looking at them directly. These three points can lead to an incorrect interpretation of the eye tracking data and therefore have to be kept in mind while analyzing the data.

Since the eye tracking data can be misinterpreted, the definition of ground interpretation methods is necessary. This leads to the following list of questions that can be answered by watching the video files and can then be used for strategy development:

1. Is the participant looking at the screen and attending to a component?
2. In which part of the interface does the participant have the most interest?
3. Does the participant use a pattern of attending to the components that can be generalized over more than one trial?

These questions are answered in Table 12 for the six participant video files. Since one trial is not significant for the overall eye movement all twenty trials were taken into account when creating the participant pattern. This tried to minimize the different problems of eye tracking accuracy described above. It was noticed that participants who only look at the light that is lit up and most right on the interface seem to check the other lights in a different way, for example with experience or knowledge about the state of the world. Another observation is that 90% of the participants attended to all interface components in the first trial before solving the task. This leads to the assumption that this behavior is part of creating a mental map of the position of every component on the interface.

Table 12. Analysis of the eye tracking data.

Participant	1.	2.	3.
S10_6	Yes	Lights	Checking the lights beginning from the left until no more lights are lit up and then checking the switch for the next components, then selecting the one that is connected to the switch and not lit up.
S12_2	Yes	lights and switches	Starting from the light that is lit up and most right, then going more right to find the faulty component. The participant did not directly look at all lights before the one that was started with.
S13_3	Yes	lights and switches	This participant used a movement pattern corresponding to the Diag model.
S14_3	Yes	lights	Checking the lights beginning from the left until no more lights are lit up and then checking the switch for the next components, then selecting the one that is connected to the switch and not lit up.
S15_5	Yes	lights and switches	No clear pattern was observable.
S16_6	Yes	lights	Checking the lights beginning from the left until no more lights are lit up and then checking the switch for the next components, then selecting the one that is connected to the switch and not lit up.

However, it has to be mentioned that the video stream had a low resolution and the white cross indicating the eye movement was not always visible because most parts of the slides were also white. Also, the interpretation of the video data is a subjective task because it depends strongly on the analyst who tries to see patterns while watching the video. The results are not supported by explicit numbers but they give an impression of the participants' behavior.

4.2.3. Reaction times

The reaction time for a trial is defined as the time between changing to the slide and clicking on a component. Thus reaction time consists of the time to solve the task, the time to move the mouse to the component, and the time for clicking on the mouse button. When analyzing the reaction times, the mouse movement and the clicking time were considered as one time. The time to solve the task is separated into slopes per cycle and the time for mouse movement. For analyzing the additional data and generating results that can support the additional study, the same methods as in section 3.2.2 and (Ritter & Bibby, 2008) were used.

First, all valid participants were analyzed together and their performance per trial was compared with the predictions per trial. In order to analyze the significance between participant performance and additional Diag model prediction linear regression was used. The linear regression resulted in a correlation of r = 0.56. The mean intercept is 7.49 and therefore is similar to the mean mouse movement time over all trials and participants with a mean mouse movement time of 6.25s. The regression coefficient is 0.0169. These results were used to calculate the average of problem solving time over participants and trials. The predicted times and the actual performance per trial are shown in Figure 21.

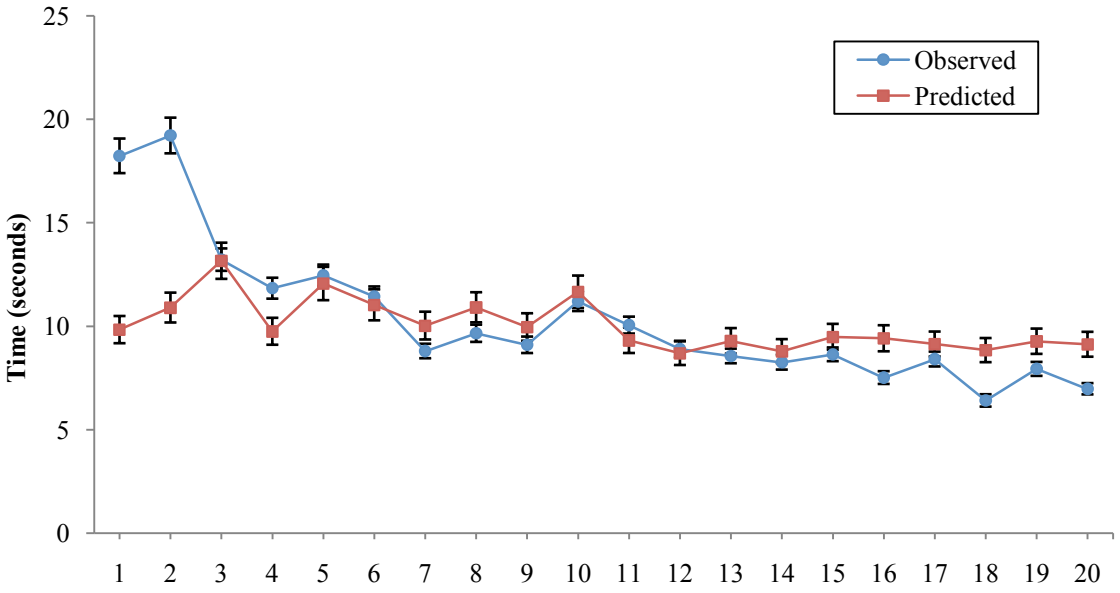


Figure 21. Predicted and observed problem-solving times (means and standard error) for trials 1 to 20 averaged over participants and trials.

The first two trials in Figure 21 differ more than the rest of the trials. The linear regression for the comparison of observed times and predicted times for 20 trials showed a correlation of r = 0.56. When reducing linear regression to trials 3 to 20 a correlation of r = 0.82 was reached. No such effect was measured in the existing user data. This might be caused by the different instruction sets because the existing instructions gave an example for the task while the additional study did not. This leads to the assumption that the participants in this particularly study used the first two trials as a task orientation phase. Within this phase the participants adapted to the type of task and might have developed a strategy for the task. Since the influence of the orientation phase should be minimized, only reaction times of trials 3 to 20 will be examined.

Table 13 shows the results of the linear regression between the observed and additional model times separated by participants.

Table 13. The participant ID, correlation (R), intercept (in seconds), regression coefficient (B, in seconds per model cycle), and number of correct responses for each participant for trials 3 to 20 when cycles are regressed onto problem-solving time.

Participant	R	Intercept	B	N
D1_7	0.51	5.41	0.04	18
D2_8	0.49	5.79	0.02	18
D3_9	0.65	5.41	0.02	18
D4_10	0.07	12.61	0.01	17
D6_2	0.19	8.70	0.01	18
D7_3	0.31	4.93	0.01	16
D8_4	0.65	3.19	0.02	18
D9_5	0.37	8.59	0.01	18
D10_6	0.29	9.65	0.02	14
D11_7	0.20	10.69	0.01	15
D12_8	0.64	3.89	0.01	18
D13_9	0.27	5.85	0.01	18
D14_10	0.69	3.81	0.01	18
D15_11	0.37	4.98	0.02	18
D16_11	0.31	12.83	0.02	18
D17_11	0.73	4.92	0.02	18
D18_11	0.28	6.38	0.01	18
D19_11	0.13	7.53	0.00	17
D20_11	0.22	9.37	0.01	16
D21_11	0.17	7.53	0.01	18
S1_3	0.50	8.68	0.04	17
S2_9	0.78	3.57	0.04	18
S3_8	0.18	8.98	0.01	18
S5_4	0.63	5.88	0.03	13
S6_2	0.14	12.26	-0.01	16
S7_5	0.63	3.57	0.01	15
S8_10	0.19	10.13	0.01	15
S9_7	0.77	2.02	0.02	18
S10_6	0.17	5.98	0.01	15
S11_1	0.17	9.67	0.01	18
S12_2	0.86	3.66	0.02	18
S13_3	0.24	6.92	0.01	17
S14_4	0.29	8.39	0.01	18
S15_5	0.45	5.53	0.01	17
S16_6	0.43	11.02	0.02	18
Mean	**0.40**	**7.10**	**0.01**	**17.09**

These results show that about 30% of the participants' behavior correlates well ($r \geq 0.5$) with the additional model predictions. This also means that more than 60% of the participants used a strategy that is different to the Diag strategy. These participants will be analyzed in chapter 5 to discover the different strategies they used. A useful preparation for this process is splitting the participants into groups with similar characteristics. Therefore, the same clustering method as in section 4.2.1 was used. In addition to the k-means algorithm a bottom-up clustering method called Agglomerative Hierarchical Clustering (AHC) was applied to determine a stable separation of the participant clusters. The AHC provided information about how many different strategies were approximately used. The

participants will be clustered into reaction time clusters. These clusters will support the strategy finding process in chapter 5 because the most efficient way to identify the strategy that a participant used is by comparing the mouse movement, eye tracking, and reaction time data. Since these data sets are not available for every participant, clustering is a useful identification method for participants that have the same strategy but use for example different mouse movement methods.

The clustering was done in two stages. At first the participants with a correlation of $r > 0.5$ were collected in cluster (1) because the prediction fit their behavior well. This does not prove that these participants used the Diag strategy; they might have shifted into it or used a strategy that produces similar behavior. They will still be left out of the strategy finding process since they might influence it in an undesired way. Cluster (1) consists of 12 participants. Second, the remaining participants were clustered with the k-means algorithm. As feature vectors for the k-means algorithm the following indicators were chosen:

- The average reaction time over trial 3 to 20 for each participant because the average reaction time depends on the strategy used by the participant and the individual participant's processing and kinetic speed.
- The correlation between the additional model predictions and the additional observed data because this value indicates how strongly the additional strategy and the actual participant strategy differ.
- The number of correct answers per strategy because they indicate how error free a strategy works.

The number of clusters for the k-means algorithm was determined by the AHC which used Ward's distance to combine the multi vector clusters. The dendrogram in Figure 22 reaches a stable state with three different clusters. Therefore the k-means algorithm was initialized with three random start points. K-mean ran 100 times and found the best inter class variance at iteration 19. The three resulting clusters and their participants are shown in Table 14 and will be used in chapter 5 for strategy finding.

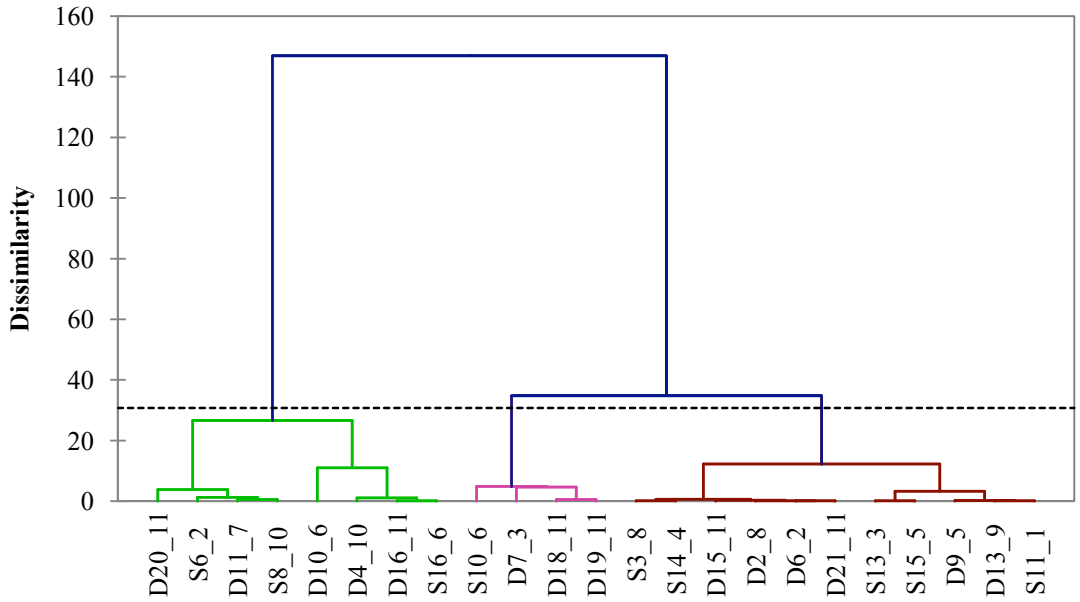


Figure 22. Dentrogram for the Agglomerative Hierarchical Clustering of the participants that did not fit the Diag prediction.

Table 14. Result of the k-mean clustering used to divide the participants into strategy groups.

Clusters	(2)	(3)	(4)
Number of Participants	10	5	8
Average Distance To The Centroid	1.481	1.575	1.367
Participants	D2_8 D6_2 D7_3 D15_11 D18_11 D19_11 D21_11 s3_8 S10_6 S14_4	D4_10 D10_6 D16_11 D20_11 S16_6	D9_5 D11_7 D13_9 s6_2 s8_10 S11_1 S13_3 S15_5

The participants in cluster (1) used strategies that fit well to the Diag model prediction. In order to support the Diag model and its predictions a detailed analysis of cluster (1) is necessary. Therefore the same procedures as in section 3.3 were chosen to measure the quality of the additional model. The average correlation for cluster (1) and the additional model predictions is 0.67. The average intercept is 4.5 s and the average slope is 227 ms per cycle. These values were used for the calculation of predictions for every participant (as predicted cycles * 227ms + 4.5 s). Figure 23 shows the comparison between the predicted and observed average times in seconds and standard errors for each fault aggregated over the participants in cluster (1). The predicted and observed times for solving the problem are increasing along the interface from left to right. The overlap between predicted and observed times correlates with r = 0.96.

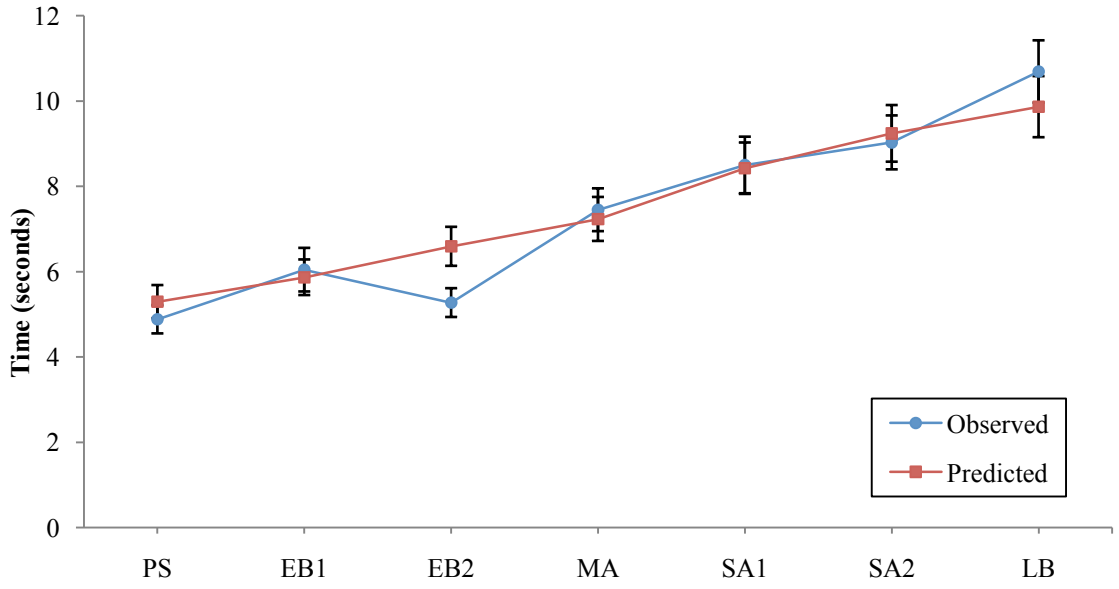


Figure 23. Comparison of the predicted and observed average times in seconds and standard errors for each fault aggregated over the participants in cluster (1).

Figure 24 shows the comparison of the predicted and observed average times in seconds and standard errors for trial 3 to 20 aggregated over the participants in cluster (1). This data shows that there is a strong relation between the predicted times and the time participants from cluster (1) took to solve the problem. The correlation is r = 0.93 and therefore the additional model provides good predictions for the learning behavior of participants from cluster (1).

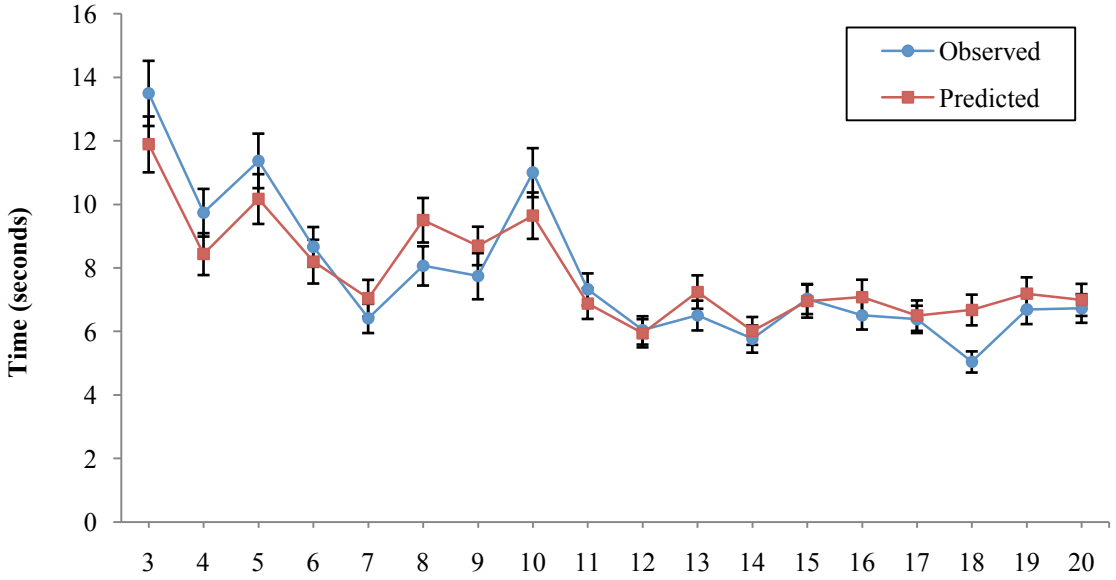


Figure 24. The observed and predicted problem solving times (means and standard errors) per fault for trials 3 to 20 averaged over cluster (1).

The third way in which the model may match the data is by comparing the average times per participant to the average times predicted by the additional model. Within the 12 participants of cluster (1) there were three pairs that saw the same order of faults (stimuli 4, 7, and 9). Figure 25 shows the average problem solving time (means and standard errors) per participant. It can be seen that participant S1_3 shows the highest deviation from the predicted time. Since this participant can be found in mouse movement group (c) this might be a result of the participant's overlap of mouse movements and thoughts.

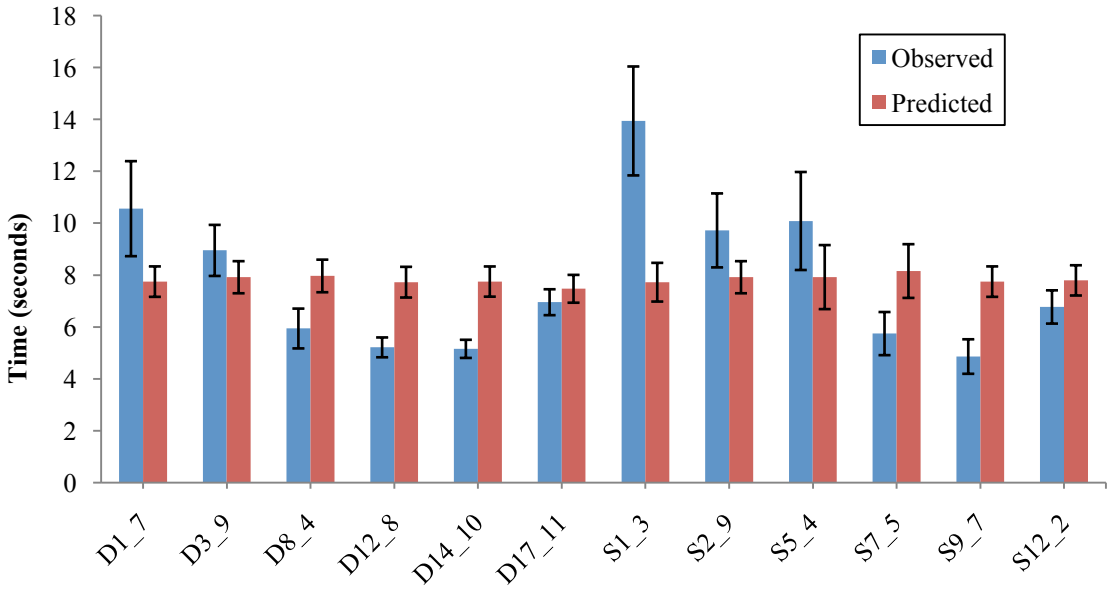
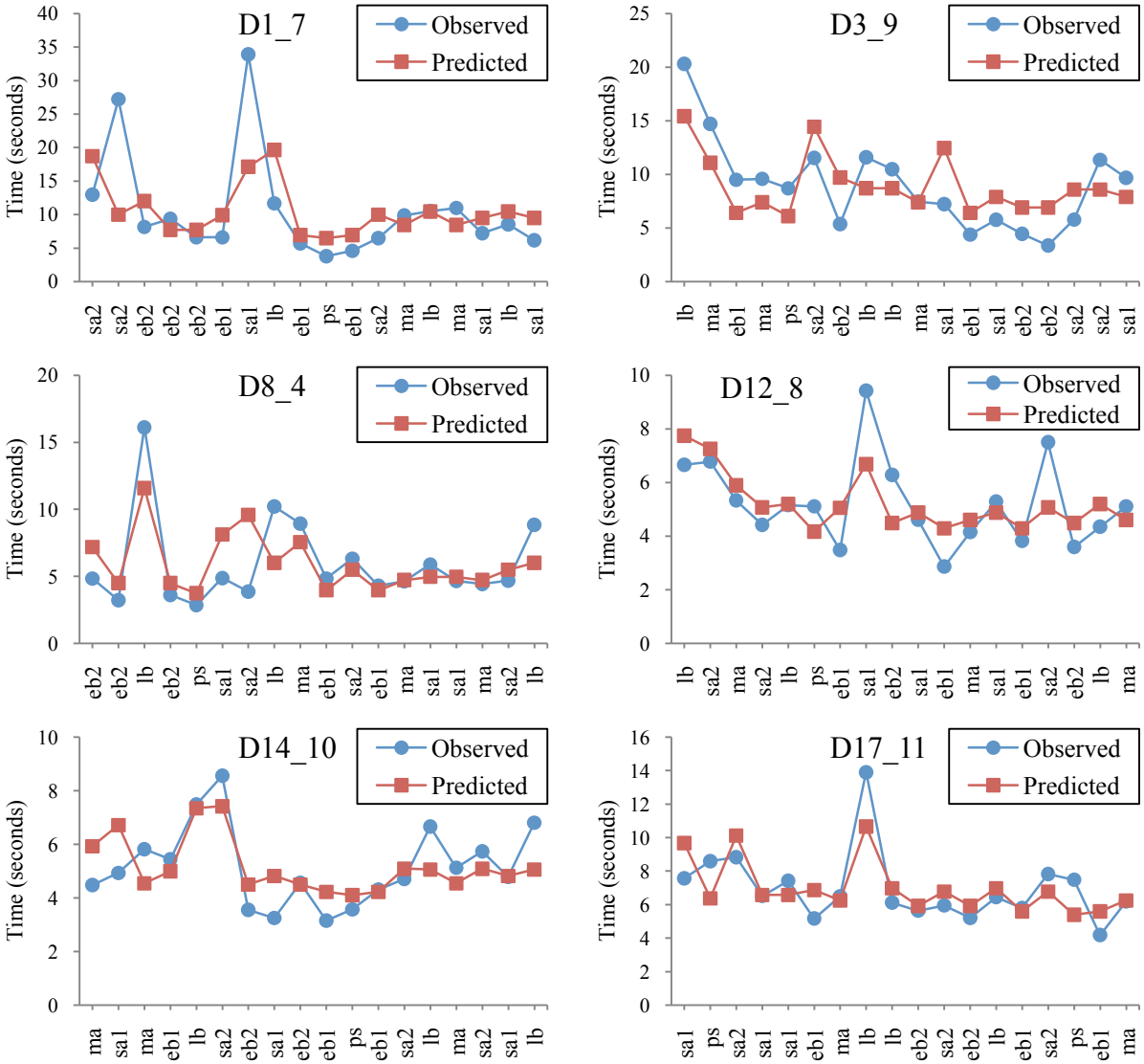


Figure 25. The observed and predicted problem solving times (means and standard errors) for the cluster (1) participants averaged over trials.

Participants in cluster (1) have a range of individual correlation to the additional model predictions from r=0.5 to r=0.8. This can have the following reasons:

- The participants had adapted the Diag strategy depending on their ability to solve problems such as the fault-finding task and therefore developed it after a different number of trials.
- The participants shifted between strategies but the Diag strategy was their primary strategy.
- The participants used a strategy that creates similar observations for the task.

However, the main statement is that the number of trials solved with the Diag strategy varies between participants. Some participants in cluster (1) show signs of a strategy shift. Figure 26 shows the individual participant performance and the individual model predictions for trial 3 to 20. For most of the trials the observations and predictions show a similar behavior. This means that strategy shifts might be responsible for the times that differ, e.g., participants D12_8 and D3_9 fitted the predictions for the first trials and then used different strategies which led to a difference between the model predictions and the observed performances.



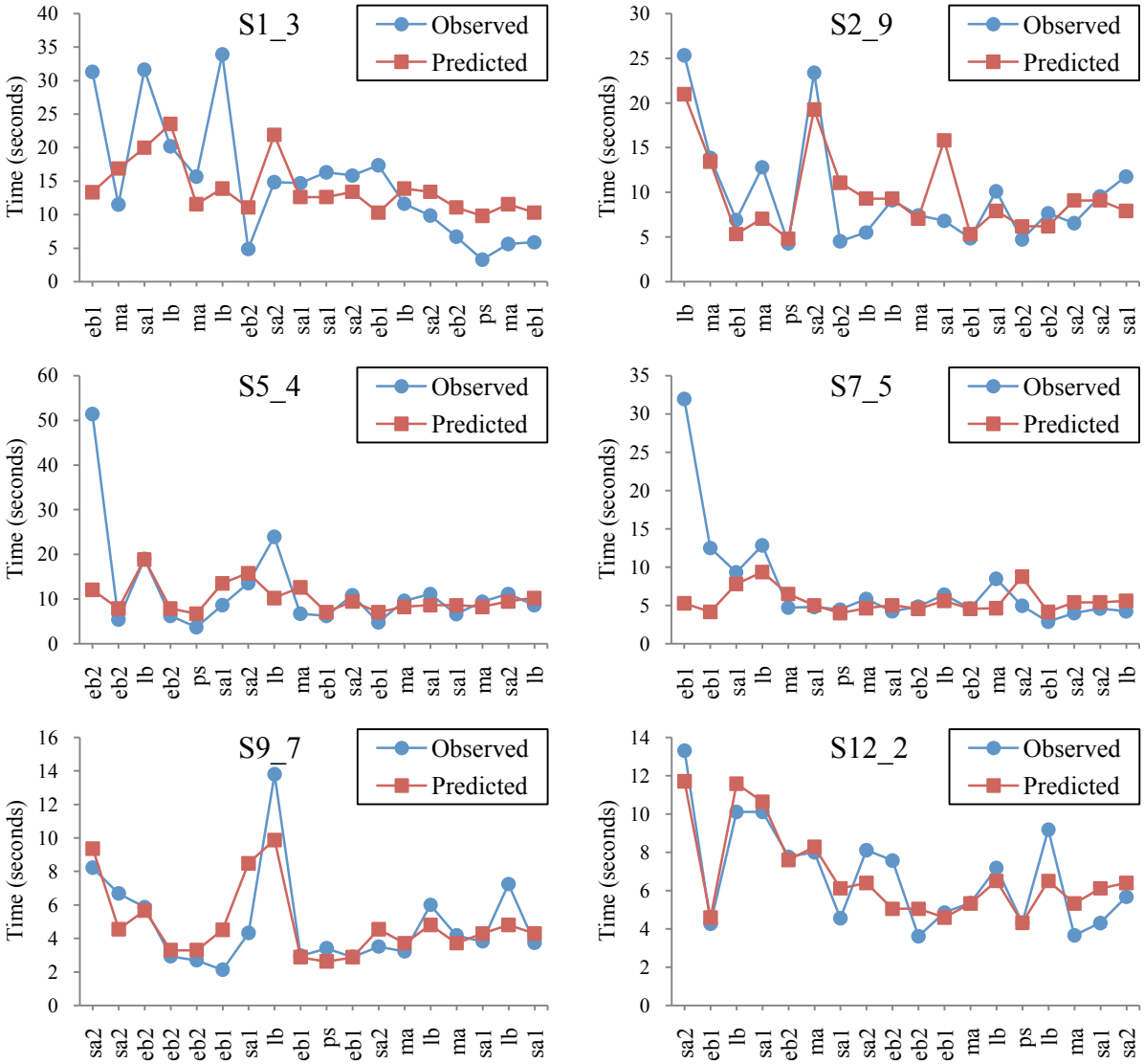


Figure 26. Predicted and observed problem solving times for the respective stimulus of cluster (1) participants.

The detailed analysis of cluster (2), (3) and (4) is not necessary because the correlation between their participants and the Diag predictions was not high enough to create meaningful results. The clusters will be used in chapter 5 to generate additional strategies.

4.3. Comparison of existing and additional Diag predictions

In order to compare the results from the existing and additional Diag study a careful analysis is necessary. There are two major differences between the two studies that need to be taken into account for an error free comparison. First, in the additional study the introductions were changed to encourage the creation of different strategies while solving the fault-finding task. Second, the additional model does not create the same predictions as the existing model. These differences led to results that differ in quality and significance.

In the existing study only two participants were found that might have had a different strategy. This led to the assumption that 80% of the participants used the Diag strategy to solve the fault-finding task. In the additional study only around 30% of the participants fitted to the additional model predictions with a correlation $r \geq 0.5$. This difference can be explained with the introduction example that was given in the existing study. With the help of this example the participants in the existing study

were able to develop a strategy before performing the task. Therefore the participants could solve the task right from the beginning. Since the additional study instructions did not have a task example the participants had to develop a strategy for the task within the first two trials.

As mentioned in section 3.2.2, the additional model and the existing model create different predictions. When the existing predictions are used for cluster (1), the average correlation between additional data and the existing predictions is r = 0.73. This means the existing predictions are more correlated in comparison to the additional model predictions that only reached a correlation of r = 0.67. These correlations are equivalent to the results in section 3.3 because the additional model predictions also do not correlate with the existing data in the same quality the existing model predictions do.

Both studies show that the average predicted and observed times over faults are a linear hierarchical growing function. This is supported by the linear equations for the predicted and observed times per faults. The equations shown in Equation 3 were created by using the least square method over the numbers in Figure 23. Both show similar hierarchical growing over the faults. This leads to the conclusion that the participants in cluster (1) needed more time when the faulty component was at the end of the circuit. The same result was found in the existing study.

Observed times per faults	Predicted times per faults
$y = 0.95x + 3.604$	$y = 0.795x + 4.315$

Equation 3. The linear hierarchical growing function per fault averaged over all participants in cluster (1).

4.4. Summary

This user study is based on the existing user study materials by Ritter and Bibby (2008). It was run at two universities, one in the United States and one in Germany. The performance of 37 participants was captured and analyzed. The main intention for running the user study was gathering additional data showing different strategy behavior. The materials, apparatus, and procedure are based on the description in Ritter and Bibby (2008) and were modified to enable a detailed analysis of additional strategies. One of these modifications was that the participants did not get an example of the fault-finding task; they had to use their mouse movement to draw their thoughts, and should answer without reflection. This led to the collection of additional data that could differ from the existing study.

Since there is no method of capturing thought, cognitive science has to rely on the observation of participants to verify its behavior models. For this user study, the participants' mouse movement, eye tracking, and reaction times were captured. All three indicators allow a view on participant behavior while solving the fault-finding task. The correlation between the additional data and the additional model prediction was used to divide the participants into a group that probably used the Diag strategy and one that did not. The 12 participants who probably used the Diag strategy supported the results of Ritter and Bibby (2008). The 23 participants who did not use the Diag strategy were divided into three clusters for a further analysis in chapter 5. The clustering was done with the k-means algorithm with feature vectors based on the average reaction times and the deviation to the Diag model predictions. These clusters were combined with mouse movement and eye tracking data to build a useful foundation for the strategy development in chapter 5.

The analysis of the additional observations shows that the additional Diag predictions did not fit them as well as the existing Diag predictions. This might be caused by the different implementations of the

existing and the additional Diag model. However, the behavior in terms of average fault times and average trial times is equal and therefore leads to the same results.

5. IMPLEMENTING NEW DIAG STRATEGIES WITH HERBAL

One result outlined in chapter 4 was that 23 participants did not use the existing Diag strategy and therefore the predicted and observed times do not correlate well. Using the k-means algorithm, these participants were divided into 3 clusters. These clusters will be used as a foundation for analyzing each participant's strategy. The development of additional strategies is necessary to analyze the different learning behaviors for the fault-finding task.

5.1. Participants who did not use the Diag strategy

Table 14 shows the participants that fit the Diag predictions with a correlation of $r < 0.5$ and therefore might have used a different strategy than the existing Diag model. These participants were separated into clusters depending on their average reaction times and the deviation to the additional Diag model predictions. The mouse movement and eye tracking results are shown in Table 10 and Table 12. They represent considerable sources of information concerning the participants' behavior. These observations and measurements were used in chapter 4 to describe patterns of individual behavior. These patterns need to be transformed into strategies that can be used to model human behavior. Therefore the strategies described in this section reflect observed behavior.

The first step for identifying a different strategy was examining the eye tracking data. Compared to mouse movement, eye tracking data is probably more accurate in representing human thoughts. Table 12 shows that there were different patterns observed when examining the eye tracking data. The behavior of a participant during the major number of the trials resulted in a pattern that can almost directly be transformed into a strategy. However, the second step was to support, generalize, or to specify these patterns with the help of mouse movement data. The mouse movement data could only be used for participants in mouse movement group (b) (Table 10). The last step was to look for similar mouse movement patterns within a reaction time cluster (Table 14). It was possible to identify participants within these clusters that showed similar mouse movement and therefore supported the found strategy patterns. These three steps led to the identification of four strategies that the participants used to solve the fault-finding task.

5.1.1. The diagram selection strategy

The diagram selection strategy (DiagSelect) is similar to the Diag strategy because it also selects and tests components stepwise. The strategy was developed based on the eye tracking and mouse movement of participants S13_3, S15_5, D8_4, D3_9 (Figure 18), and partially on the participants from reaction time cluster (2). When analyzing a participant it is possible that not the complete strategy can be seen when watching the eye tracking or mouse movements because the learning process might have led to step skipping. Therefore the strategy was designed to fit the problem solving and the learning process as well.

DiagSelect finds the faults using the Diagram knowledge, working from PS to LB. First the Diagram knowledge is used to select a component. Second, the selected component is tested. For example, when a new trial was presented to the participants, their first step was to examine SWITCH1. If SWITCH1 was ON, the participants selected PS for testing. Within this strategy the test procedure is reduced to the CHECK-LIT test. If the component is lit up, it is not faulty. Otherwise the faulty component is found. If a component is not faulty the diagram knowledge is used to find the next

switch in the circuit. The next step is attending to the selected switch to identify the component it is pointing to. This procedure of switch selection and component testing is repeated until the faulty component is found.

5.1.2. The check lights and then switch strategy

The check lights and then switch strategy (CheckLitSwitch) is based on the separation of attending to the lights and switches. The strategy was developed based on the eye tracking and mouse movements of participants S1_3, S9_7, S12_2. As mentioned for the DiagSelect strategy, when analyzing a participant it is possible that not the complete strategy can be seen when watching the eye tracking or mouse movements because the learning process might have led to step skipping. Therefore the strategy was designed to fit the problem solving and the learning process as well.

CheckLitSwitch finds the faults in two major steps. In the first step the interface knowledge is used to select the last component that is lit up by attending to the lights on the interface from left to right. Within this process every light is checked for being the last one that is lit up. The first step stops with the component that is lit up and most right on the interface. The second step uses the diagram knowledge to find the next switch in the circuit. This fault identifier switch is attended to and its state is determined. The component that the fault identifier switch is pointing to is checked. It is not lit up and the faulty component is found.

The problem when developing this strategy was to determine if the participants did check the last component before clicking on it. Theoretically they did not have to because the strategy provides the faulty component by checking the state of the fault identifier switch. Practically the observation of the mouse movements (and eye tracking data from S12_2) shows that after checking the fault identifier switch the participant moved the mouse in an almost straight line to the faulty component. However, measuring the time for this movement and the time of the eye attending to the faulty component shows that the participants did not click right away. A plausible explanation for this observation is that the participants did check if the component is lit up and therefore if it really is the faulty one. CheckLitSwitch compensates this behavior by performing a light check on the component. Another problem is that this strategy needs to assume that the participants used the full range of view to determine that the light they chose is indeed the most right.

5.1.3. The lit indicator strategy

The lit indicator strategy (LitIndicator) is similar to the CheckLitSwitch strategy and also based on the separation of attending to the lights and switches. The strategy was developed based on the eye tracking and mouse movement data of participants S10_6, D16_11, and partially of the participants from reaction time cluster (3). The strategy can be described as an optimized variation of CheckLitSwitch.

LitIndicator finds the faulty component in two major steps. First, by attending to the interface knowledge from right to left the first component that is lit up is selected. Within this process the first light that has a different state than the previous one gets selected. After the first stage the strategy changes its direction and uses the diagram knowledge to find the next switch in the circuit. This fault identifier switch is attended to and its state is determined. The component that the fault identifier switch is pointing to is checked. It is not lit up and the faulty component is found.

Here the same problem as described for CheckLitSwitch occurred. The observation described for CheckLitSwitch was also found in participant S10_6 and therefore the same explanation provided for CheckLitSwitch can be applied.

5.1.4. The previous state strategy

The previous state strategy (PrevState) solves the fault-finding task similar to the Diag strategy. The development of this strategy is based on the mouse movement of participants S19_11, S13_9, S8_10 (Figure 17), and partially of the participants from reaction time cluster (4). This strategy has never been observed for more than 5 trials in a row and therefore seems to be an experimental strategy. PrevState uses the interface and diagrammatic knowledge as separate lists of information with the first and last element linked together. This way the knowledge bases are rejoined at the position that was used last.

The first step within this strategy is to solve the first trial with the Diag strategy. As a second step the faulty component of the first trial is stored, e.g., MA. The task moves to the next trial and the third step is the determination of the proceeding direction for the new trial. The participants decided this depending on the position of the light that was lit up and most right. If this position was right to the stored component, participants would proceed from left to right. If this position was left to the stored component, they would proceed from right to left. PrevState does this by subtracting the interface position of the light that is lit up and most right from the interface position of the stored faulty component. If the result is negative, the strategy proceeds from left to right on the interface, otherwise from right to left. This way the strategy always knows in which direction the faulty component will be found. The fourth step tests the components in the determined direction, beginning with the stored component from the previous trial. After solving the current trial the previously stored component is replaced by the current faulty component. Steps two to four are repeated until all 20 trials are solved.

PrevState has the advantage of quickly identifying the faulty component if the previous and current fault are the same. However, a difficulty is the complex process of testing from right to left on the interface or backwards through the circuit because it cannot be determined whether a component is faulty without testing the previous component. All three Diag tests (light test, switch test, and previous test) are used. Since the algorithm is moving backwards the previous component has not been tested yet. Therefore the current component cannot be diagnosed. Solving this problem is only possible by using the diagrammatic knowledge and the previous light. This means if PrevState is moving from right to left on the interface the test for the previous component is modified by using the diagrammatic knowledge and checking the light of the previous component. If the previous component is lit up the faulty component is the one that is currently being checked.

5.2. Implementation of the different strategy models

The strategies described in section 5.1 were implemented into separate Herbal models. Depending on the strategy, every model creates predictions of participant behavior while solving the fault-finding task. In order to generate predictions that fit well to actual human behavior it was necessary to work accurately when transforming the descriptions of the strategies into problem space models. The predictions could also be used to improve the models by adjusting the strategy where its predictions do not correlate well.

The implementation of the strategy models was done in 4 steps. First, the new model structures were developed to identify which problem spaces, operators, actions, conditions, and types were required. As described in section 2.3.2, components of a Herbal model are reusable in a different model. Therefore the second step was to analyze the components from step 1 in order to identify elements from the additional model that can be reused. Third, the actual implementation of the model was done using additional model elements and newly created elements. As the additional model the strategy models are based on the working memory type described in section 3.1.2. Since Herbal compiles into Soar 8 the strategy models also work on the same learning principles that are applied to the additional model. The fourth step was the adaptation of the Java environment to the strategy. This was necessary because every strategy has different requirements to the environment, e.g., the order of the knowledge bases or the way of accessing the knowledge. The implementation of the strategy models was done in three days per strategy depending on the amount of new operators and problem spaces. Within this process steps 1 and 3 were the most time-consuming.

5.2.1. DiagSelect

The structure of the DiagSelect model is shown in Figure 27. The model is less complex than the additional model and solves the fault-finding task with fewer steps. It consists of 5 problem spaces and 13 operators. The Herbal compiler transformed this structure into 50 Soar rules. Since there are a lot of elements that were reused from the additional model, the description of DiagSelect is reduced to the operators that have a different functionality.

DiagSelect uses the DIAGNOSE and FIND-FAULT problem space like the additional model. The first change was made to the CHOOSE-COMPONENT operator. Within this strategy, it addresses the DIAGRAM-SELECT problem space directly. After DiagSelect enters the DIAGRAM-SELECT problem space it chooses the next component that is in the diagram knowledge base. In two cases this results in two answers (PS → EB1 & EB2; EB2 → SA1 & SA2) and therefore the model has to attend to the responsible switch for checking which component is connected. This is done by entering the CHECK-WORLD problem space and then switching back to the DIAGNOSE problem space to attend to the responsible switch. The connected component is selected for testing and therefore the TEST-COMPONENT problem space is entered. Testing in the TEST-COMPONENT problem space is reduced to the CHECK-LIT test. The CHECK-PREVIOUS and CHECK-SWITCH tests were not implemented because the strategy already covers these cases. If the component is lit up, the RESET operator in the FAULT-FINDING problem space fires and the next component is selected for testing. If the component is not lit up, the REPORT operator in the DIAGRAM problem space fires and the current faulty component is found. This procedure is repeated until the stimulus is solved.

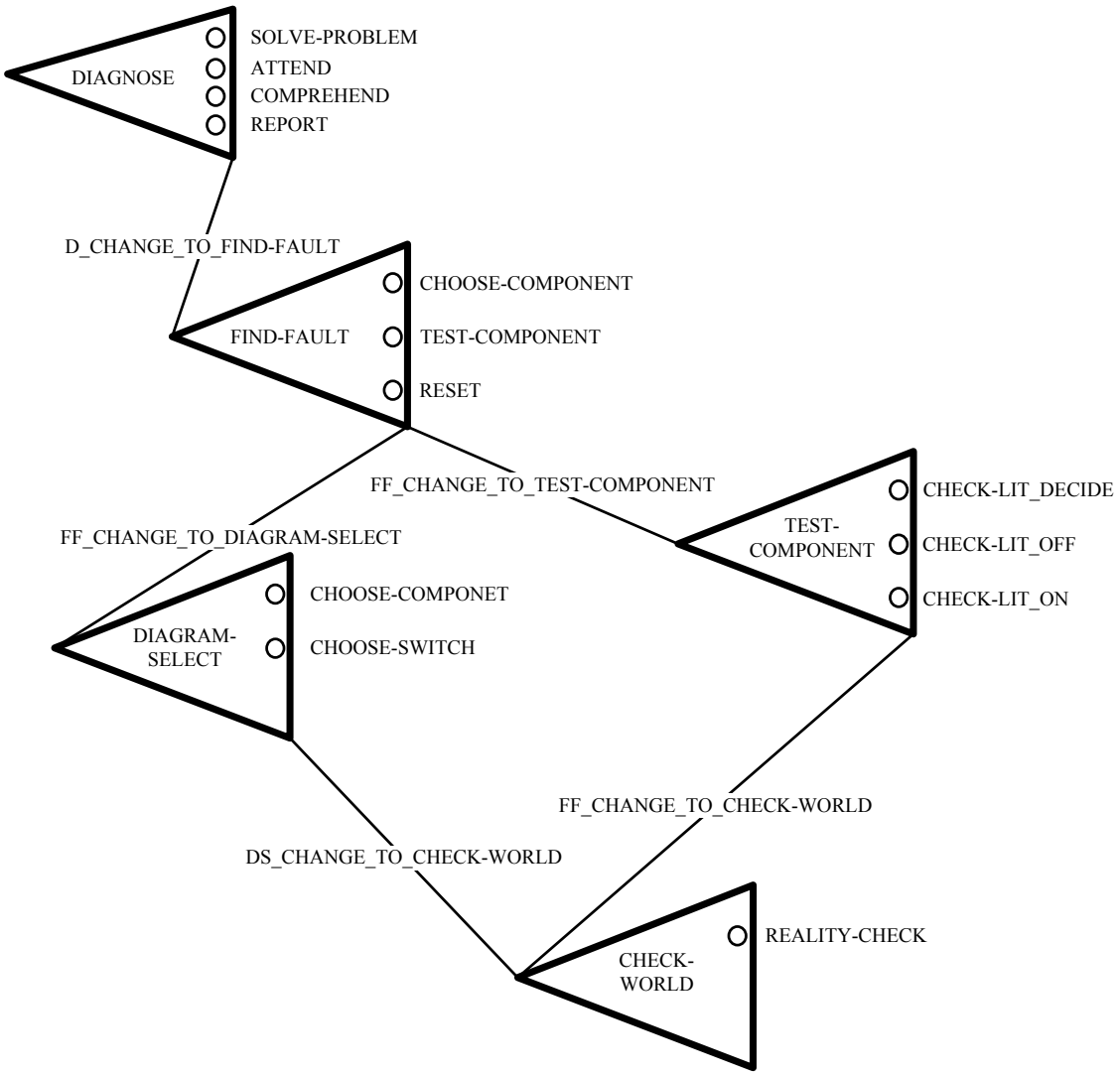


Figure 27. The problem space structure of the DiagSelect model.

5.2.2. CheckLitSwitch

Figure 28 shows the structure of the CheckLitSwitch model. It consists of 6 problem spaces and 14 operators. The Herbal compiler transformed this structure into 55 Soar rules. How this strategy solves the fault-finding task is described in section 5.1.2. This description is the basis for the order in which CheckLitSwitch fires its operators. In order to understand the model's behavior when solving the task an introduction to the operators is necessary.

CheckLitSwitch uses the DIAGNOSE problem space to initialize the model state and define its goals. The fault-finding process begins with the identification of the "last-lit-component". This component is defined as being lit up and most right on the interface. Therefore the CHECK-INTERFACE problem space is used. This problem space begins by attending to PS and checking if it is the "last-lit-component". Then it works its way through the interface from left to right. The lights are tested by using the REALITY-CHECK operator to change into the DIAGNOSE problem space. Within the DIAGNOSE problem space the ATTEND operator fires. If the "last-lit-component" was not found, the COMPREHEND_CI operator fires. The model changes to the CHECK-INTERFACE problem space and selects the next interface light. If the "last-lit-component" was found the COMPREHEND_FF operator fires and the model changes to the FIND-FAULT problem space. After this transition the model behaves equally to the DiagSelect model. It selects the next component in the

circuit by attending to the next switch. This switch is evaluated by changing into the CHECK-WORLD and DIAGNOSE problem spaces. Then the selected component is tested in the TEST-COMPONENT problem space whether it is lit up. If not, it is reported as faulty.

For CheckLitSwitch the Java environment was changed to meet the models' requirements. The request operators were adapted and the interface knowledge base was extended by the information if a light is the "last-lit-component" or not. This implies that the model does not explain how the participants acquire the information about the "last-lit-component".

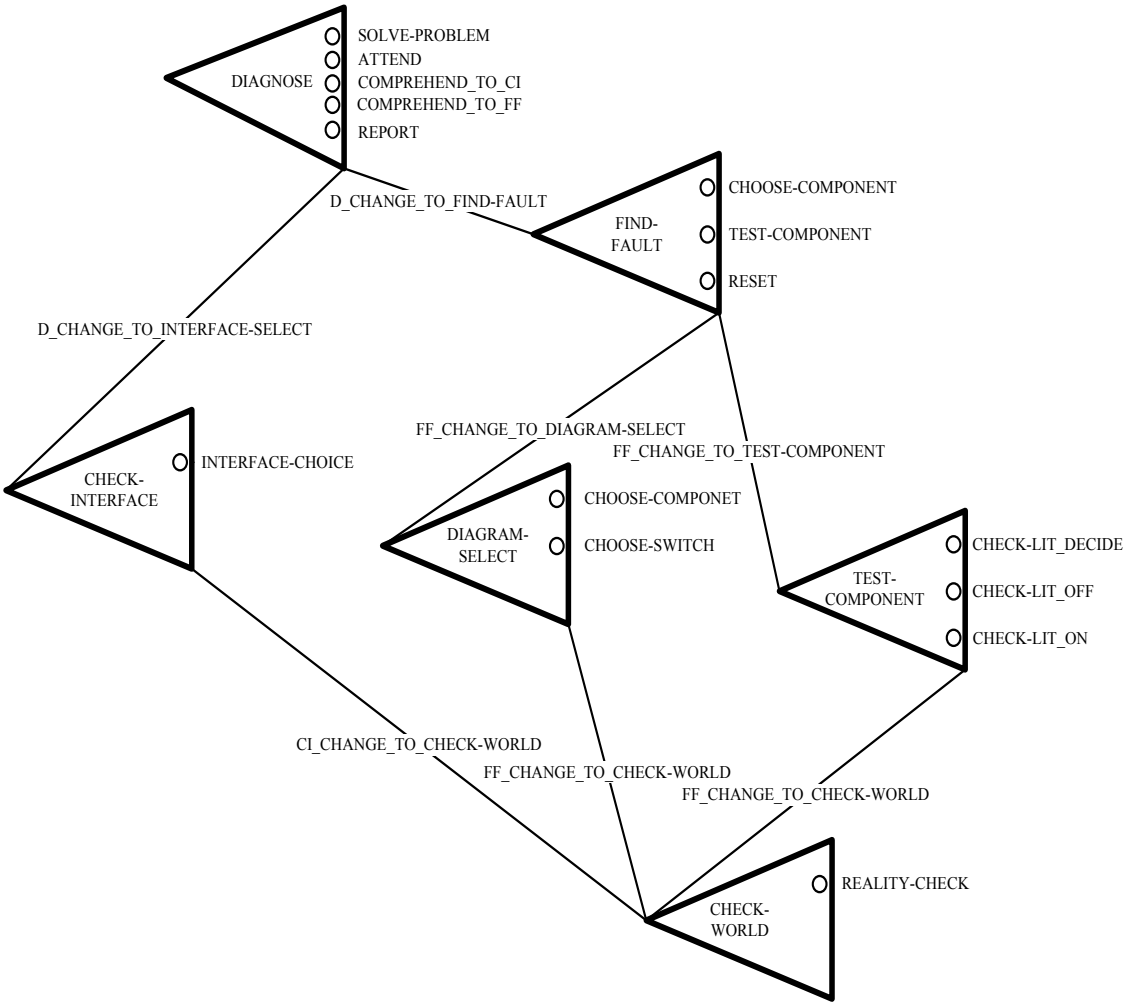


Figure 28. The problem space structure of the CheckLitSwitch model.

5.2.3. LitIndicator

The structure of the LitIndicator model is shown in Figure 29. It consists of 6 problem spaces and 14 operators. The Herbal compiler transformed this structure into 45 Soar rules. For a better understanding of the LitIndicator model a short description of its basic operators and their order is necessary.

LitIndicator uses the DIAGNOSE problem space to reset the model state and define its goals. The fault-finding process begins with the identification of the "last-lit-component", which has been described in section 5.2.2. Since the participants attended to the interface from right to left (LB to PS), the process of identifying the "last-lit-component" is faster. The model is taking this into account by using the CHECK-INTERFACE problem space more directly without attending to the CHECK-

WORLD problem space first. When the “last-lit-component” is found the COMPREHEND_FF operator fires and the model changes to the FIND-FAULT problem space. LitIndicator selects the next component in the circuit by firing the operators in the DIAGRAM-SELECTION problem space. The switch responsible for the next component is evaluated by changing into the CHECK-WORLD and DIAGNOSE problem spaces. Then the selected component is tested in the TEST-COMPONENT problem space whether it is lit up. If not, it is reported as faulty.

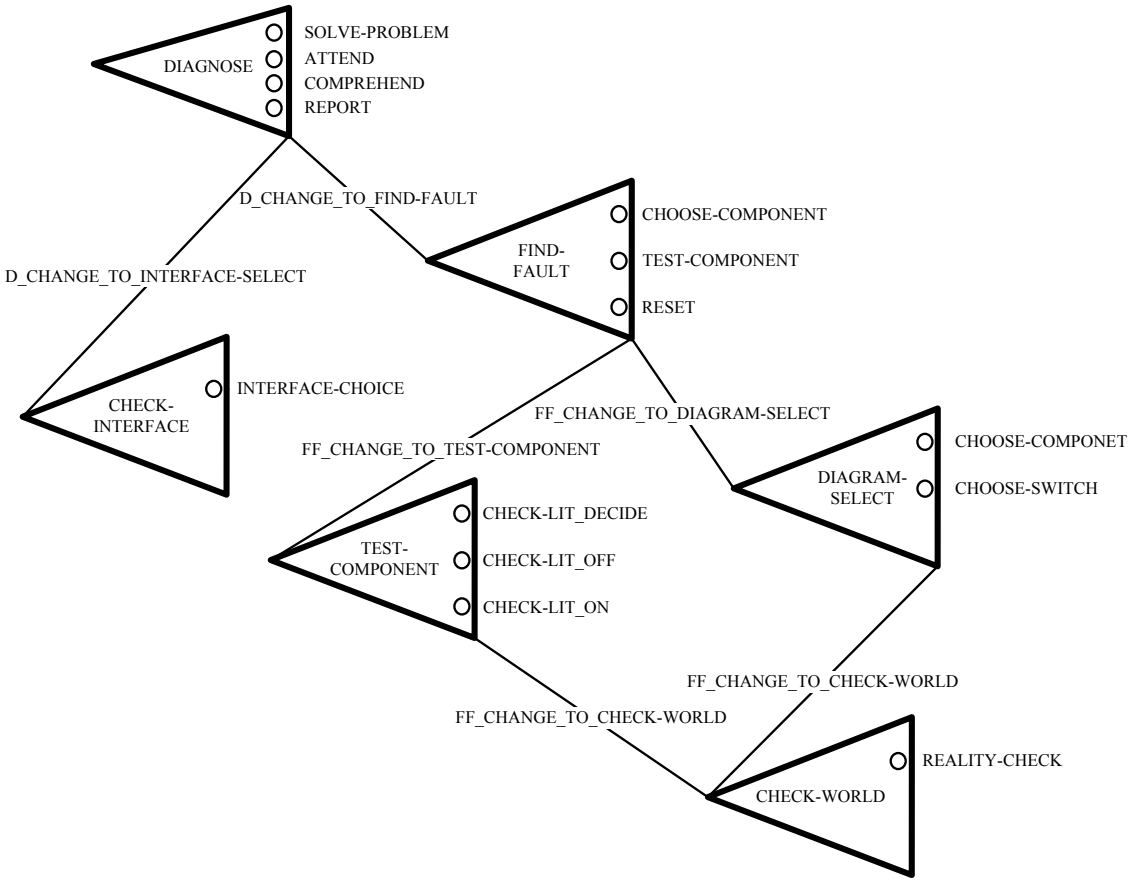


Figure 29. The problem space structure of the LitIndicator model.

5.2.4. PrevState

Figure 30 shows the structure of the PrevState model divided into problem spaces and operators. PrevState consists of 7 problem spaces and 21 operators. The Herbal compiler transformed this structure into 80 Soar rules. The PrevState model tests components by using the same order of operators as the Diag model but its knowledge base depends on the direction of the problem solving process. For a better understanding of the PrevState model a short description of its basic operators and their order is necessary.

PrevState uses the DIAGNOSE problem space to reset the model state and define its goals. PrevState begins by changing into the SET-DIRECTION problem space. It uses the DIRECTION-DECIDE operator to request the knowledge that is needed to determine the direction on the interface. When this knowledge is gathered the decision about the direction is made by the CHOOSE-NORMAL (left to right) or CHOOSE-BACKWARDS (right to left) operator. After this decision process the model changes into the FIND-FAULT problem space and starts checking the component that was faulty in the previous trial. If this component is not faulty, the model starts selecting and testing all components

in the chosen direction. This is done with the same order of operators that the Diag model uses. As soon as the faulty component is found the model reports it by firing the REPORT operator.

Out of all four strategy models, PrevState made the most considerable changes to the Java environment necessary. First, the environment has to store the faulty component from the previous trial. Second, when the direction is changed the knowledge bases concerning the task and the order of components have to be reversed. For example, if the previous fault was LB and the model used the normal direction, starting with LB the model would work backwards and therefore the interface and diagram knowledge would have to be reversed. In this case the interface knowledge would provide SA2 as the next component. Third, when the model is working backwards it cannot determine if the current component is faulty because the previous check has not been run yet. In this case the model would determine the previous component based on the diagram knowledge and attend to the light status of this component. If it is lit up, the current component is faulty.

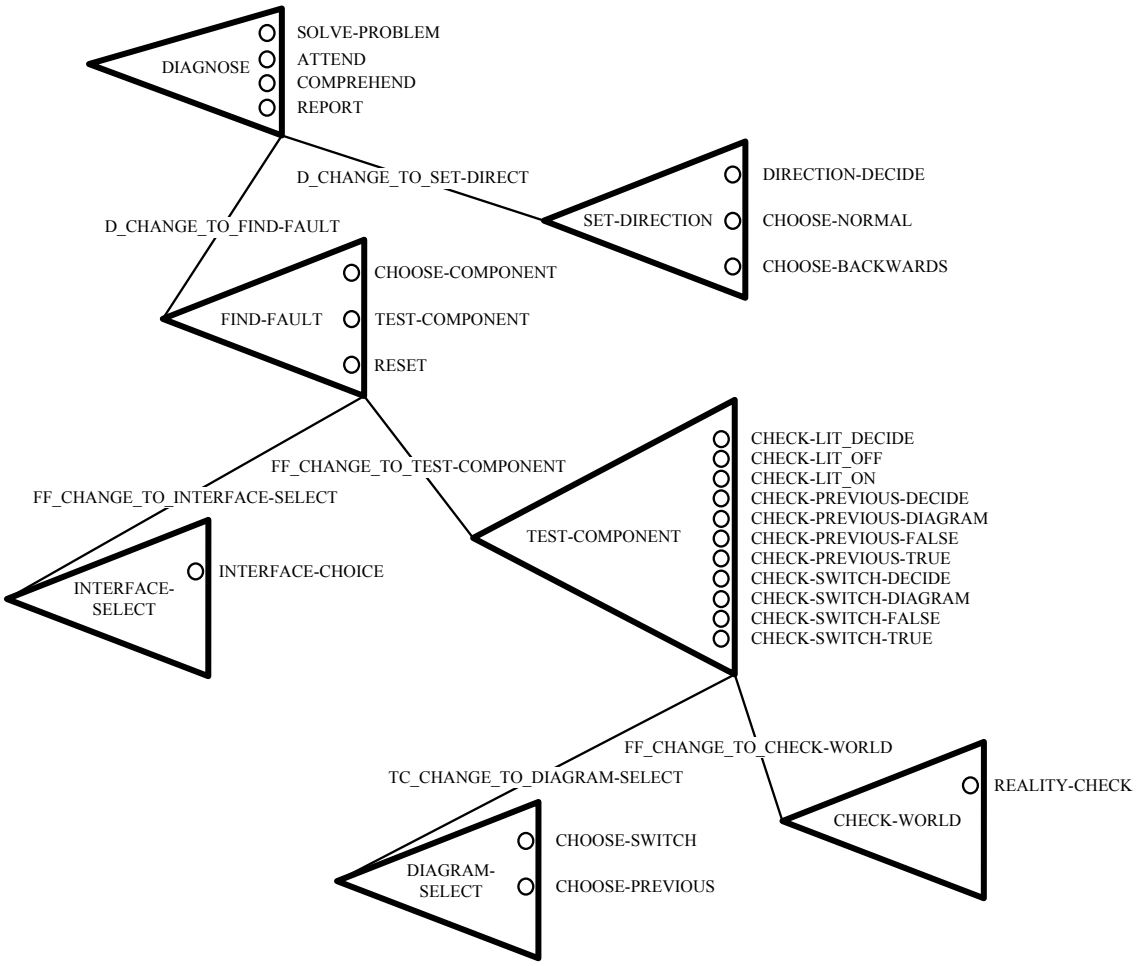


Figure 30. The problem space structure of the PrevState model.

5.3. Comparison of strategies to the user data

The strategies developed in section 5.1 have to be evaluated in order to determine their potential for predicting human performance. Therefore the implemented strategy models were used to solve stimuli 1 to 11. This means that for every stimulus five prediction sets (Diag model predictions + 4 strategy model predictions) were created. The prediction sets were compared to the gathered user data from chapter 4 to find the best fit per participant. The participants that did not correspond well or at all were analyzed for their possible use of multiple strategies. Also the question was raised which reaction time

cluster used which strategy and whether there is a connection between study major and chosen strategy.

Initially the prediction sets were correlated with the data of the 35 valid participants. Since the existing and additional study used different methodologies, the non corresponding participants of the existing study were not used for this comparison. The resulting correlations per participant and strategy were then filtered to identify the strategy with the highest correlation per participant. Table 15 shows the participants and the strategy they correspond best. This way the participants were categorized into 5 strategy groups.

Table 15. Participant, their strategy, correlation (R), intercept (in seconds), regression coefficient (B, in seconds per model cycle) when participants are regressed to the best corresponding strategy.

Participant	Strategy	R	Intercept	B
S12_2	Diag	0.86	3.66	0.021
D14_10		0.69	3.81	0.009
D12_8		0.64	3.89	0.009
D1_7		0.51	5.41	0.036
D8_4	DiagSelect	0.884	-0.932	0.149
D3_9		0.810	1.321	0.166
S5_4		0.757	0.903	0.209
S15_5		0.677	2.181	0.109
D15_11		0.564	0.564	0.155
D18_11		0.530	4.344	0.065
D9_5		0.526	5.636	0.100
S13_3		0.475	2.430	0.130
D6_2		0.409	4.921	0.109
D11_7		0.380	5.548	0.169
D10_6		0.363	3.937	0.197
S14_4		0.356	6.205	0.086
S11_1		0.349	5.678	0.116
D21_11		0.338	5.125	0.075
S9_7	CheckLitSwitch	0.854	-0.282	0.138
S2_9		0.786	0.523	0.242
D17_11		0.779	3.424	0.095
S7_5		0.754	1.766	0.101
S1_3		0.566	4.422	0.253
D2_8		0.507	4.146	0.127
S16_6		0.450	9.127	0.121
S10_6	LitIndicator	0.549	-1.185	0.163
D16_11		0.438	9.140	0.110
D20_11		0.375	5.282	0.107
S8_10		0.347	6.148	0.101
S6_2		0.306	14.811	-0.065
S3_8		0.258	7.787	0.042
D19_11	PrevState	0.618	12.050	-0.037
D13_9		0.611	1.635	0.055
D4_10		0.451	4.861	0.094
D7_3		0.372	7.697	-0.020

First, Table 15 suggests that participants developed different strategies to solve the task. It can be seen that each strategy group consists of at least four participants and the highest correlation within each

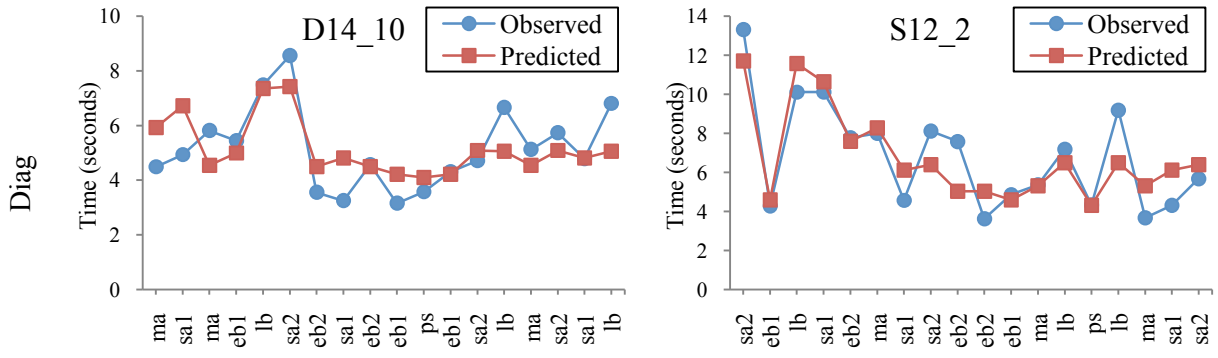
group is $r \geq 0.5$. Most correlations are not significant but show that the participant probably used parts of the respective strategy. Second, the number of participants in the different strategy groups varies from 4 to 14 which means that some strategies are used more often than others. This fact could be caused by several circumstances such as the intelligence quotient of the participants, experience with similar tasks, or the ability to concentrate on the task. Third, as in the existing study there was participant behavior that could not be predicted. These participants might have used a different strategy which has not been discovered within this thesis. For further research on the three suggestions mentioned, the following sections will discuss the behavior of the best corresponding participants, possible strategy shifts of participants that did not correspond to a single strategy, and participants that did not correspond at all.

In order to analyze whether the participants might have training in reading diagrams and schematics their majors were examined (Appendix 5 – Participants and their majors). It was examined for all participants whether the major and the choice of strategy correlate. This led to the result that half of the participants with a mechanical engineering major chose the DiagSelect strategy; the other half chose the CheckLitSwitch strategy. Every other major group used each strategy at least once and mostly with an equal distribution among all available strategies.

5.3.1. Participants who fit well to a strategy

Participants that fit well to a strategy are defined by a correlation of $r \geq 0.70$. Applying this condition to Table 15, 9 participants remain and therefore will be analyzed in the following section. Figure 31 presents their individual performance in separate plots to see how accurate the prediction follows the observed performance. This section also takes into account the results of the analysis of cluster (1) done in section 4.2.3. Participants that were analyzed there reached a higher correlation with different strategies than the Diag strategy.

Figure 31 shows the observed and predicted behavior for the participants that corresponded well to a strategy. The strategy was regressed for trials 3 to 20 for each participant individually. The graphs show that the predicted times match the observations well. This strongly supports the assumption that participants chose different strategies to solve the fault-finding task. It can also be seen that participants S7_5 and S5_4 took longer to develop their strategy than the other participants because their third (and fourth) trial took around five times longer than the predicted values. This shows that the number of trials that is necessary for finding a strategy differs for each participant. Since the performance of S5_4 and S7_5 fits to the predicted behavior for trials 4 (5) to 20, they probably needed four trials to develop to their strategy.



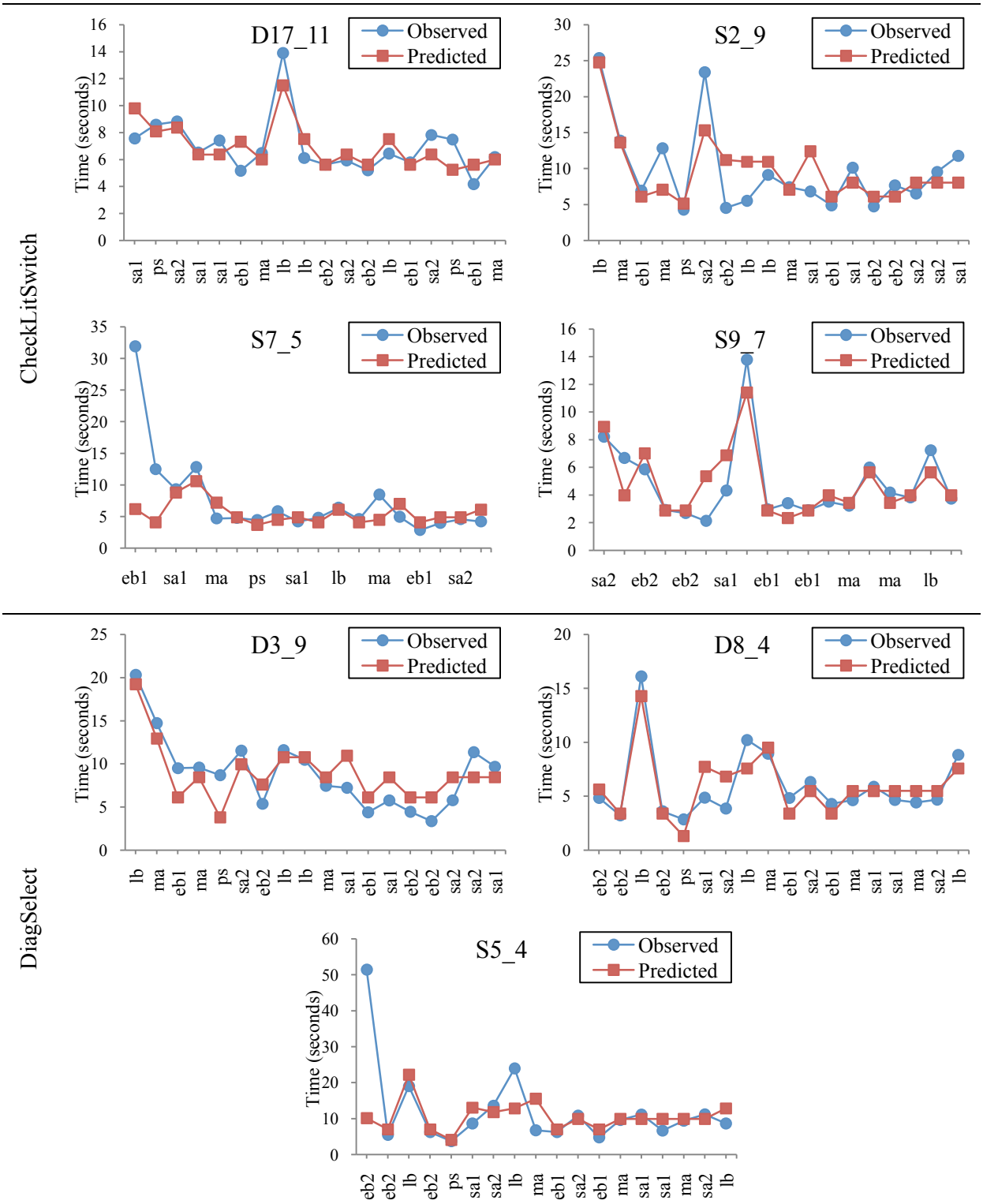


Figure 31 Predicted and observed problem solving times in task presentation order for participants with a correlation of r ≥ 0.7 for the individual strategy.

6 out of the 10 participants from cluster (1) did correlate better with a different strategy than the Diag strategy and therefore were assigned to a new strategy group. This reassignment was necessary because of the low ratio of correlation that was chosen in section 4.2.3 to categorize cluster (1). Also it has been noticed that the difference between the correlations of different strategies per participant was sometimes smaller than 0.05. For example, S2_9 correlated to the CheckLitSwitch strategy with r = 0.786 and to the Diag strategy with r = 0.78. The comparison of the intercept values from linear

regression (Diag Intercept = 3.57s; CheckLitSwitch Intercept = 0.523s), and the fact that S2_9 is in mouse movement group (a) (group with the fastest mouse movement and clicking time; section 4.2.1) results in the conclusion that the CheckLitSwitch strategy describes the performance of S2_9 better. Similar conclusions were drawn for the other participants.

Only 9 participants correspond well enough to a strategy that the individual prediction indicates that they used only this strategy. Therefore an analysis of the remaining participants that might have used several strategies within the 20 trials is necessary.

5.3.2. Participants who switched between strategies

The identification of participants that used more than one strategy to solve the fault-finding task is difficult because there are no direct indicators that prove a strategy shift. Therefore the combination of the two best fitting strategies per participant is the most promising indirect indicator for a strategy shift. This process started with filtering the data in Table 15. The participants from section 5.3.1 were left out because they fitted well to a single strategy. Table 19 (Appendix 6) shows the remaining participants and their two best fitting strategies.

The participants from Table 19 were analyzed with a strategy shift identification algorithm. This algorithm tries to apply the two best corresponding strategies to the stimulus data and thereby to find the highest correlation for the observed times and the combined prediction times. Combined prediction times consist of a set of prediction times that was created by using one strategy for the first part of the stimulus and the other strategy for the second part of the stimulus. The algorithm finds the highest correlation by stepwise moving the border between the two parts of the stimulus.

The algorithm begins by assigning the better correlated strategy to trials 3 to 5. The second strategy is assigned to trials 6 to 20. Therefore the first border is set at trial 6. Trials 1 and 2 were excluded because of their training character. In addition, it was defined that a strategy has to cover at least 3 trials. The individual trial prediction times were calculated by using the individual intercept and regression coefficient of the model cycles and the observation. After the combined times were determined, linear regression was used to calculate the correlation between the observed and combined times. Then the border between the two parts of the stimulus was moved stepwise up until it reached trial 18. The correlation was calculated for each border position. Within this process the following rules were checked to create valid results:

1. All participants reduced their mouse movement and clicking time while solving the stimulus. Since the intercept value is an indicator for the mouse movement and clicking time, it has to be higher for the first than for the second strategy.
2. The regression coefficient had to lie between 0.03s and 0.30s because Newell intended that a Soar model cycle should lie within a range of 30 and 300 ms (Newell, 1990, p. 121 & 224).
3. If the correlation between observed and combined times is higher than for the previous border position, the new border position is stored.

If one of these conditions could not be fulfilled, the maximum correlation was determined without them. Since the strategy shift can occur in both orders, the next step was to restart the algorithm with the reversed order of strategies. Table 16 shows the results of the strategy shift identification algorithm for the participants from Table 19.

Table 16. Participant ID, border (in trial), first strategy selected, second strategy selected, condition 1 (intercept for first strategy > intercept for second strategy), condition 2 (0.03 < regression coefficient of both strategies < 0.3), and the maximum correlation between the combined and the observed times.

Participant	Border	First Strategy	Second Strategy	Condition 1	Condition 2	Maximum R
S10_6	7	DiagSelect	LitIndicator	True	True	0.94
D12_8	10	Diag	LitIndicator	True	False	0.75
S15_5	9	DiagSelect	CheckLitSwitch	True	True	0.75
D4_10	8	CheckLitSwitch	PrevState	True	False	0.72
D19_11	14	PrevState	DiagSelect	True	False	0.72
D13_9	10	PrevState	CheckLitSwitch	True	True	0.70
D20_11	7	Diag	LitIndicator	True	False	0.68
S6_2	16	LitIndicator	PrevState	True	False	0.68
D9_5	13	CheckLitSwitch	DiagSelect	True	True	0.67
S11_1	16	PrevState	DiagSelect	True	True	0.65
D1_7	9	Diag	DiagSelect	True	True	0.64
D7_3	14	PrevState	DiagSelect	True	False	0.63
S1_3	9	CheckLitSwitch	DiagSelect	True	True	0.61
D6_2	7	PrevState	DiagSelect	True	False	0.60
D15_11	11	DiagSelect	CheckLitSwitch	True	True	0.60
D16_11	13	Diag	LitIndicator	True	False	0.60
D18_11	8	PrevState	DiagSelect	True	False	0.60
D2_8	11	CheckLitSwitch	Diag	True	True	0.58
S14_4	6	Diag	DiagSelect	True	False	0.53
S13_3	14	DiagSelect	CheckLitSwitch	False	True	0.48
S16_6	10	CheckLitSwitch	Diag	True	True	0.45
D21_11	9	CheckLitSwitch	DiagSelect	True	True	0.42
D10_6	13	CheckLitSwitch	DiagSelect	True	True	0.40
S8_10	14	LitIndicator	PrevState	True	True	0.39
D11_7	14	PrevState	DiagSelect	True	True	0.38
S3_8	6	CheckLitSwitch	LitIndicator	True	True	0.38

The results presented in Table 16 show that participants S10_6, S15_5, and D13_9 fulfilled both conditions and the correlation between their observed and combined times reaches $r \geq 0.7$. This proves that these participants used two strategies to solve the stimuli. For these three participants, the border between the two strategies lies between the 7^{th} and 10^{th} trial. This means it is below the average value of 10.6 for all participants from Table 16. Figure 32 shows the observed times and the combined times from the strategy shift identification algorithm for the three participants. These plots show that the combined times match the observed times well which illustrates that these participants switched between strategies.

For the participants from Table 16 with $r < 0.7$ there is no definite strategy classification. The low correlation might be caused by the fact that the decision for changing a strategy and the process of adapting to the new strategy takes time. Therefore the predicted and observed times do not correlate well during the shifting process. However, an analysis of the order of strategies in Table 16 was done and the results are presented in Table 17.

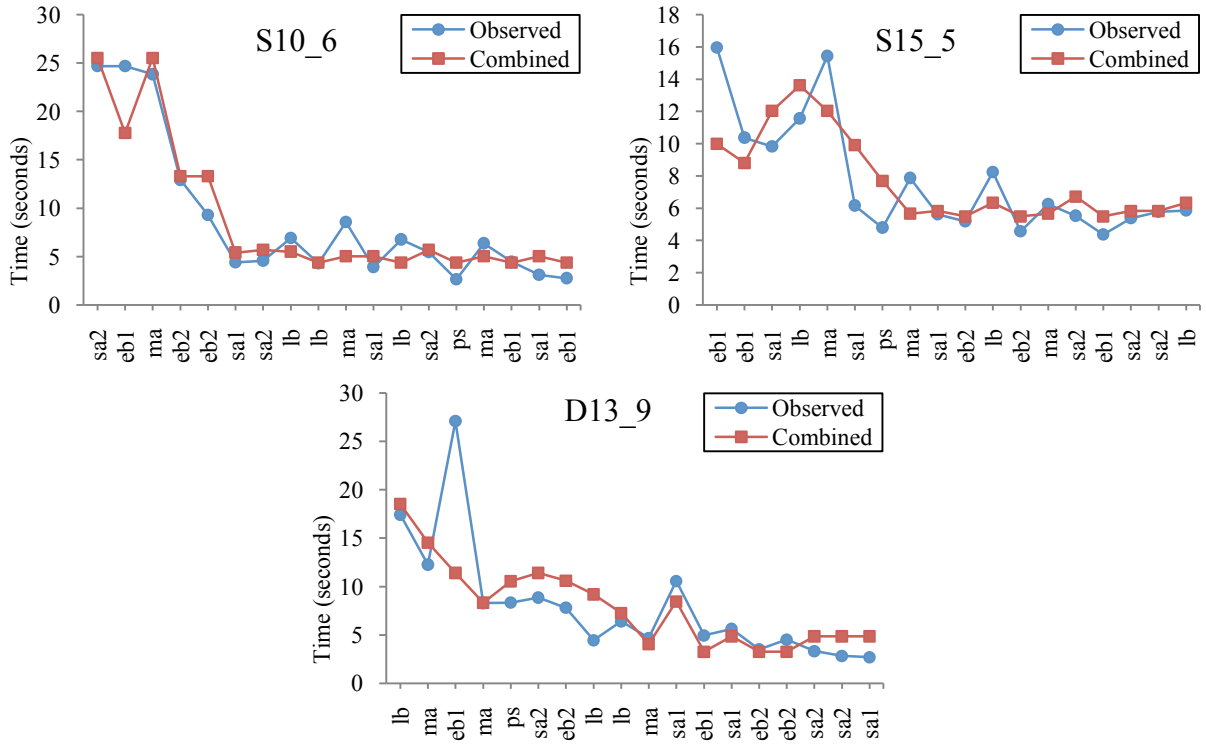


Figure 32. Participants who switched between two strategies with their observed and combined times from the strategy shift analysis algorithm.

Table 17. Strategy name, quantity how often it was used as first and second strategy, and special observations from the analysis.

Strategy	Used first	Used second	Special observations
Diag	5	2	When Diag was the first strategy, the participants only switched to LitIndicator or DiagSelect.
DiagSelect	4	12	46% of the participants switched from any of the other strategies to the DiagSelect strategy.
CheckLitSwitch	8	4	When CheckLitSwitch was the second strategy, 75% of the participants had switched from DiagSelect.
LitIndicator	2	5	The average border value when LitIndicator was used as second strategy is 8.6 and therefore 2 trials below the average of 10.6.
PrevState	7	3	86% of the participants that used PrevState as their first strategy switched to DiagSelect.

The observations from Table 17 show that the participants tend to switch from a strategy with a high memory allocation and several tests to a less complex strategy. The special observations can be interpreted as follows:

(a) The special observations on Diag, CheckLitSwitch, and PrevState indicate that the transition is done by dividing the original strategy into subtasks. An example for a subtask is a light check. Then one subtask is chosen and adapted to cover the fault-finding task requirements.

(b) The high percentage of participants that switched to DiagSelect shows that participants tend to use their diagrammatic knowledge more often if they had tested it in other strategies before.

Besides the possibility to switch between two strategies only, participants might have also tried more strategies or switched back and forth. This behavior could not be identified by the strategy shift identification algorithm, and therefore cannot be excluded as a reason for lower correlations.

5.4. Summary

In chapter 5, four strategies for the fault finding task could be developed based on the analysis of participant behavior. For making the strategies as accurate as possible the mouse movement, eye tracking, and reaction time clusters from chapter 4 were used. The strategies were mapped onto a problem space structure. This structure was implemented in Herbal to create models that can predict human behavior while solving the fault-finding task. These models consist of reused components from the additional Diag implementation from chapter 3 and newly developed components. Every strategy model was run to predict the number of model cycles for each of the 11 stimuli. Human performance for these stimuli had been captured in the additional user study (chapter 4).

The predictions from the additional Diag model and the four new strategy models were analyzed for their correspondence to the participant observations. It could be shown that participants vary in the strategies they choose to solve the fault finding task. Out of the 35 valid participants from chapter 4, 9 participants could be assigned to one single strategy. These participants reached a correlation of $r \geq 0.7$ between the observed times and the individual predicted times of the strategy. It could also be shown that these participants used different strategies to solve the fault finding task. This means if no strategy is provided, the participants develop different strategies to solve the fault-finding task. The analysis of the participants' study major suggests that the kind of chosen strategy depends on the possible knowledge about similar tasks. For example, participants with a mechanical engineering background developed only two different strategies.

In the process of assigning the participants to the strategy that created the best predictions for their behavior, every strategy group reached at least four members. However, LitIndicator and PrevState could not predict participant behavior with a correlation above 0.7. These strategies start by assuming knowledge which the participants might still gather while solving a number of trials. For example, the LitIndicator model knows that the faulty component is always behind the most right and lit up component on the interface. The PrevState strategy assumes that the two knowledge bases can be used forwards and backwards. Therefore the knowledge necessary for both of these strategies was theoretically available for the participants, but they had to analyze the task carefully enough while reading the instructions. Neither the mouse movement nor the eye tracking data supported that possibility. As a result it can be assumed that LitIndicator and PrevState might have been used as a second strategy after the participants had gone through some trials and learned about the task.

The participants with observed behavior that could not be predicted by the strategy models were analyzed for strategy shifts. This means their two best corresponding strategy predictions were combined to create the maximum correlation between the observed and combined times. A strategy shift identification algorithm was used to determine the highest possible correlation. Under the restriction of plausible conditions the algorithm moved the border between the two strategies to find the best corresponding distribution. Three participants could be identified that fulfilled the conditions and had a correlation of $r \geq 0.7$ between the observed and combined times. Therefore these participants used two strategies to solve the task. The final analysis attended to the order of strategies of all participants that did not fit to a single strategy. Some behavior patterns concerning the way participants switched between strategies could be shown.

After evaluating the additional user data with the available strategies some participants still cannot be assigned to a strategy or a strategy shift. This can be explained as follows:

- The participants were not fully concentrated when doing the task and therefore their observed times vary strongly. For example, some participants almost showed no signs of learning.
- The participants might have shifted between strategies more than once. This could not be analyzed because only 20 trials were captured and therefore not enough data was available to define 3 sub stimuli that correlate with three strategies.
- The participants might have shifted between strategies within one trial to find the strategy they want to use for the rest of the trial.
- It is also possible that more strategies were developed but not identified within this thesis.

These are the reasons why only 40 % of the additional participants could be covered with the predictions made by the additional models. In comparison to the 80% in the existing study this is an interesting result and shows that a lot of potential is undiscovered in the research about developing cognitive skills.

6. CONCLUSIONS

In conclusion, this thesis makes a contribution to the development of cognitive models in a high level language, the problem solving strategy development, the implementation of strategies in this high level language, and their comparison to observed behavior. This was done by using a fault-finding task. The task was solved by humans within a user study and by a cognitive model. The observations and predictions were compared and showed that the model was able to predict the behavior of participants who used the same or similar strategies as the model. To get a detailed view of the strategies that were used in the user study to solve the fault-finding task four additional cognitive strategy models were developed. Depending on the correlation between observed human performance and model predictions it was classified which of the five strategies (Diag, DiagSelect, CheckLitSwitch, LitIndicator, PrevState) was used by whom. This way it was possible to clearly assign 12 performances to one or two strategies.

This thesis has shown that humans develop different strategies for the same type of problem when given the same training and training sequences. When solving the same set of problems repeatedly, humans learn to use these strategies more effectively and thus decrease their performance time across trials. By implementing these strategies in a cognitive architecture with a learning mechanism the learning that occurs on an individual, trial-by-trial level could be predicted. This allows a view of the human ability to solve problems by developing strategies and how these strategies are optimized during learning. In addition, it was possible to show the emergence of strategy shifts when analyzing different strategies. In some cases, the combined predictions of two strategies matched the observed behavior better than the prediction of a single strategy.

Furthermore, a new cognitive language, Herbal, was used to implement the models described in this thesis. Thus, it was possible to create a model on an abstract level that is in terms of the PSCM. The models that were implemented in Herbal were compiled into Soar code. The resulting Soar models use a single learning mechanism. In this thesis, the Soar learning mechanism has proven itself capable of performing a task using a model created with Herbal. The model also matched the user data at a detailed level, including the rate of individuals learning over the series of 20 trials. The high correlation with the data can be best explained with the model's ability to acquire and transfer different types of knowledge rather than simply focusing on one kind of learning.

To implement strategy models in other cognitive languages, the models, not just those that change strategies, need to have the ability to interact with a task or a simulation of the task. This requires the model to have a variety of knowledge, where to look, and how to choose which component to examine next. The new strategy models might duplicate the results presented in this thesis. When the same results are created, this data would help modelers to differentiate learning mechanisms and other architectural constraints. One step in this direction can be seen in the Jess code because Herbal also compiles its models into Jess. Unfortunately, Jess does not support the same kind of learning as Soar and therefore would create different predictions, but it shows how the models are implemented in a different language.

The research on strategy creation and strategy shifts can be used not only in the field of cognitive psychology but also in human computer interaction (HCI). Because interfaces have to be designed for the most common strategies and strategy shifts that are performed by their users. This way interfaces, especially for security and safety reasons, can be designed to encourage behavior that results in a minimum number of possible errors.

6.1. Comparison to the Diag paper results

The paper of Ritter and Bibby (2008) was used as a basis for this thesis and therefore a comparison of the results is useful to see in which regards they are consistent. The main difference is the percentage of participants that used the Diag strategy. Ritter and Bibby (2008) concluded that around 80% of the predictions of the existing Diag model correlated well with participant performance. In this thesis the additional Diag model was able to predict only 11% of the participants with a well fitting correlation. The following reasons might have led to this difference:

- Changes in the methodology of the study were made, especially on the interface and the given introduction. For example, the software for presenting the interface was changed and graphically updated to meet participant expectations of an interface with lights (lit up lights were drawn yellow). The most influential change to the tutorial materials was the removal of a task example. This way the participants were encouraged to develop their own strategies. This led to a range of different strategies that were captured and analyzed.
- An eye tracking system was used for some of the participants. Even if this measuring apparatus was designed to not influence the participants it might have made the participants unsecure and therefore changed their behavior.
- The additional model does not learn at the same pace as the existing model. Thus not all participants that might have used the Diag strategy were identified.

Because the difference between both models are significant the results from this thesis cannot be compared directly to Ritter and Bibby (2008). Despite the difference there are findings that both studies support. First, both show that the Diag strategy is used by some of the participants and that the use of individual data and a cognitive model that learns helps to examine how cognitive skills develop. Second, it is shown that not all participants use the Diag strategy. There were two participants in the existing study that did not fit to the Diag predictions. In the additional study it could be shown that participants develop a range of different strategies to solve the fault-finding task. In contrast to the results of Ritter and Bibby (2008) these additional strategies seem to cover a wider range of the human behavior than expected.

6.2. Model developing with Herbal

A major part of this thesis was the development of cognitive models in the high level cognitive modeling language Herbal. The process of model development can be divided into two phases. In the first phase the existing Diag model was reimplemented. Within this phase four models with different abilities were created. This was necessary because in Herbal there is no guideline for the implementation of models that are as big as Diag. The models had to fulfill the following conditions:

- supporting the Soar learning mechanism;
- keeping a consistent view of the environment;
- a good reusability of model components by using abstract names and actions that are flexible to changes.

Several Diag models were created in Herbal to analyze the design patterns. Only one out of the four models fit these conditions well enough to proceed and create additional model predictions. The first phase took about 6 weeks and was done with the idea of reusing most of the work in the second phase. The second phase was the implementation of the strategies that were observed in the user study. These

strategies were implemented in Herbal to predict participant behavior based on their strategy and the order of problems. For this process the strategies were mapped on a PSCM structure to simplify the model implementation in Herbal. Each strategy took only three days to implement because a lot of additional model components and the PSCM structure could be reused.

Since Herbal took such a central role in this thesis a statement can be made to help future cognitive modelers with their decision whether to use Herbal. Herbal and its graphical user interface allow an easy beginning in cognitive modeling. The Herbal menus have a clear structure and therefore are easy to understand and memorable. Since Herbal uses the PSCM structure it provides the modeler with a good overview of the programming progress that can be seen at the number of operators, conditions, and actions. With a few clicks Herbal modelers can create Soar programs, change parts of them, or reorganize its structure.

However, Herbal still has some design flaws that were described in section 3.1.4. While using Herbal some bugs were discovered that made the work with Herbal more difficult than it had to be. The Herbal language completely fulfilled the needs of this thesis and when the mentioned bugs have been removed it can be a useful research tool in cognitive science.

6.3. Fault-finding strategies

The strategies Diag, DiagSelect, CheckLitSwitch, LitIndicator, and PrevState were used to predict human behavior. The Diag strategy was provided by Ritter and Bibby (2008) and the others were developed based on observations. The strategies were designed to model the aspects of learning and problem solving in human behavior. They solve the fault-finding task correctly but with individual numbers of necessary sub-steps. This means the faulty component is always identified with differing efforts per strategy. The results from section 5.3 indicate that it is likely that not all strategies that were performed could be identified.

People tend to use those strategies that are easy to combine with their physical understanding. It can be assumed that people use knowledge they already have to create strategies for problems. Because people know that energy flows from the source to the apparatus, they develop strategies based on this fact. For example, four out of five strategies do not check the components backwards through the circuit or from right to left on the interface. Therefore their ability to create new strategies is restricted by the knowledge they already have about the physical world.

In section 5.3.2 it could be shown that participants shift between strategies. This was proven by applying different strategies to the same observation in order to maximize the correlation of combined predictions. Strategy shifts did not occur often. Therefore it could only be guessed what the motivation for a strategy shift was. An analysis of the strategy combinations shows that both strategies had similar parts. The found combinations support the assumption that strategy shifts are done to reduce the cognitive activity by changing to a strategy that can be executed with less checking or that is easier to learn.

Transferring the methodology of this thesis to the Tower of Hanoi task it would be expected to reproduce the same results. By implementing the three possible strategies for the Tower of Hanoi into Herbal models the effects of learning could be modeled. This way a detailed analysis of the participant behavior and thoughts would be possible. Knowing that there are only three documented strategies for the Tower of Hanoi it can be stated that the predicted data should reach a higher correlation to the observed data.

6.4. Future research implications

The research conducted by Ritter and Bibby (2008) served as a basis for this thesis and could be extended with additional data and results. Besides, the results of this thesis facilitate further research in some fields such as automatic or semi-automatic model development and self-reflection. For both of these fields the results can be a foundation for interesting new research in cognitive science. Also, new methodologies for the work on strategies and strategy shifts have been demonstrated.

Ritter (1993) pointed out that using automatic or semi-automatic model development could help generating cognitive decision models. These decision models and the strategy models in this thesis have a lot in common and can be extended in two directions. First, the process of solving the fault-finding task could be transformed into a semi-automatic process based on the interface view and an agent. The agent would have to be trained to solve the process and could communicate with humans by asking what to do next. These questions must allow every action on the interface. The human would develop a strategy by maneuvering the agent through the task. This way the strategy used to train the agent would be documented in the protocol data of the agent.

Second, an automatic method for developing, testing, and comparing strategies to the observed data could be developed. By analyzing the strategy models it could be shown that all of them include a process for selecting components and testing them. By dividing all strategies into subparts it might be possible to identify only a few basic operators or orders of operators. An automatic process could reorganize these subparts to generate every strategy possible and then compare it to user data. This way the optimal strategy for every observed data could be calculated. The workload of such a project is kept to a minimum because the basic operators can be modeled in Herbal and then stored. A program could build the model files and Herbal would compile them into Soar files. These files could be loaded automatically into a Soar kernel to create predictions. Since the Herbal model files are based on XML the best programming language for the project would be Java because it could organize the subparts, compile the model, initialize the Soar kernel, and evaluate the results.

Self-reflection research could benefit from the different strategy models as well. As the review pointed out, Diag does not support any kind of self-reflection. Since self-reflection plays an important role in human strategy development and problem solving it should also be implemented in a model. This could be done by using a set of strategy models and switching between them while doing a set of fault-finding tasks. After each task an evaluation could be implemented. The strategy would be changed if the last strategy did not fit a given requirement. In case a certain strategy fulfills the needed requirements right from the beginning, the Diag model could choose, learn, and optimize this strategy.

REFERENCES

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

Anderson, J. R. (1983). *The architecture of cognition*. Cambridge, MA: Harvard University Press.

Anderson, J. R. (1993). *Rules of the mind*. Hillsdale, NJ: Erlbaum.

Anderson, J. R. (1995). *Learning and memory*. New York, NY: John Wiley and Sons.

Anderson, J. R. (1996). *Cognitive psychology and its implications* (3rd ed.). New York, NY: W. H. Freeman.

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

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

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.

Blessing, S. B., & Anderson, J. R. (1996). How people learn to skip steps. *Journal of Experimental Psychology: Learning, Memory, and Cognition, 22*, 576-598.

Bourne, L. E., Jr., Healy, A. F., Rickard, T., & Parker, J. (1997). Strategies, strategy transitions, and the strategic basis of performance. *Manuscript submitted for publication.*

Bransford, J., Brown, A. L., & Cocking, R. R. (1999). *How people learn : brain, mind, experience, and school*. Washington, D.C.: National Academy Press.

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

Charness, N. (1991). Expertise in chess: The balance between knowledge and search. In K. A. Ericsson & J. Smith (Eds.), *Studies of expertise : Prospects and limits*. Cambridge (UK): Cambridge University Press.

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

Cohen, M. A., Ritter, F. E., & Bhandarkar, D. (2007). *A Gentle Introduction to Herbal* (Tech. Report No. ACS 2007 - 1): Applied Cognitive Science Lab, School of Information Sciences and Technology, Penn State. acs.ist.psu.edu/acs-lab/reports/cohenR04.pdf.

Cohen, M. A., Ritter, F. E., & Haynes, S. R. (2005). Herbal: A high-level language and development environment for developing cognitive models in Soar. In *Proceedings of the 14th Conference on Behavior Representation in Modeling and Simulation,* 133-140. 05-BRIMS-043. Orlando, FL: U. of Central Florida.

Cooper, R., & Shallice, T. (1995). Soar and the case for unified theories of cognition. *Cognition, 55*, 34.

Crutcher, R. J. (1989). The Role of Mediation in Knowledge Acquisition and Retention: Learning Foreign Vocabulary Using the Keyword Method. 86.

Daudelin, M. W. (1994). *Learning from experience through reflection.* Unpublished Thesis (Ed D), Boston University, 1994.

Davidson, J. E., & Sternberg, R. J. (2003). *The Psychology of Problem Solving*: Cambridge University Press.

Davis, L. W., & Ritter, F. (1987). Schedule optimization with probabilistic search. In *The Third IEEE Computer Society Conference on Artificial Intelligence Applications,* 231-236. Washington, DC: IEEE Press.

Delaney, P. F., Reder, L. M., Staszewski, J. J., & Ritter, F. E. (1998). The strategy specific nature of improvement: The power law applies by strategy within task. *Psychological Science, 9*(1), 1-8.

Evertsz, R., Ritter, F. E., Russell, S., & Shepherdson, D. (in press). Modeling rules of engagement in computer generated forces. In *Proceedings of the 16th Conference on Behavior*

Representation in Modeling and Simulation, 07-BRIMS-021. Norfolk, VA: U. of Central Florida.
Feigenbaum, E. A., & Simon, H. A. (1984). EPAM-like models of recognition and learning. *Cognitive Science, 8*, 305-336.
Firby, R. J. (1989). Adaptive Execution in Complex Dynamic Worlds [microform].
Fitts, P. M. (1964). Perceptual-motor skill learning. In A. W. Melton (Ed.), *Categories of human learning* (Vol. 47, pp. 381-391). New York: Academic Press.
Friedman-Hill, E. (2003). *JESS in action*. Greenwich, CT: Manning Publications.
Friedrich, M., Cohen, M. A., & Ritter, F. E. (2007). *A Gentle Introduction to XML within Herbal* (Tech. Report): Applied Cognitive Science Lab, School of Information Sciences and Technology, Penn State. acs.ist.psu.edu/acs-lab/reports/cohenR04.pdf.
Geary, D. C., & Brown, S. C. (1991). Cognitive Addition: Strategy Choice and Speed-of-Processing Differences in Gifted, Normal, and Mathematically Disabled Children. *Developmental Psychology, 27*(3).
Geary, D. C., & Wiley, J. G. (1991). Cognitive Addition: Strategy Choice and Speed-of-Processing Differences in Young and Elderly Adults. *Psychology and Aging, 6*(3).
Gonzalez, C., Lerch, J. F., & Lebiere, C. (2003). Instance-based learning in dynamic decision making. *Cognitive Science, 27*, 591-635.
Graham, P. (2003). Beating the Averages. *This article is derived from a talk given at the 2001 Franz Developer Symposium*.
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., & Bovair, S. (1984). The role of a mental model in learning how to operator a device. *Cognitive Science, 8*, 255-273.
Laird, J. E. (2003). The Soar Tutorial (Version www.eecs.umich.edu/~soar/getting-started.html).
Laird, J. E., & Congdon, C. B. (2005). *Soar User's Manual: Version 8.6* (Technical No. CSE-TR-72-90): Electrical Engineering and Computer Science Department, University of Michigan.
Laird, J. E., Newell, A., & Rosenbloom, P. S. (1987). Soar: An architecture for general intelligence. *Artificial Intelligence, 33*(1), 1-64.
Larkin, J. H., McDermott, J., Simon, D. P., & Simon, H. A. (1980). Expert and novice performance in solving physics problems. *Science, 208*, 1335-1342.
Lehman, Laird, & Rosenbloom. (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.
Lüdtke, A., Cavallo, A., Christophe, L., Civaldi, M., Fabbri, M., & Javaux, D. (2006). Human Error Analysis based on a Cognitive Architecture. *Proceedings of the International Conference on Human-Computer Interaction in Aeronautics (HCI-Aero 06)*.
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., Shaw, J. C., & Simon, H. A. (1958). Elements of a theory of human problem solving. *Psychological Review, 65*, 151-166.
Newell, A., & Simon, H. A. (1972). *Human problem solving*. Englewood Cliffs, NJ: Prentice-Hall.
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. (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.

Reder, L. M., & Ritter, F. E. (1992). What determines initial feeling of knowing? Familiarity with question terms, not the answer. *Journal of Experimental Psychology : Learning, Memory & Cognition, 18*(3), 435-451.
Ritter, F. (2008). RUI-Recording User Input. from http://acs.ist.psu.edu/projects/RUI/
Ritter, F. E. (1993). *TBPA: A methodology and software environment for testing process models' sequential predictions with protocols* (Technical Report No. CMU-CS-93-101): School of Computer Science, Carnegie Mellon University, Pittsburgh, PA.
Ritter, F. E., & Avraamides, M. (2001). Testing the Situation Awareness Panel. In *Proceedings of the 21th Soar Workshop,* 35-39. AI Lab, U. of Michigan.
Ritter, F. E., & Bibby, P. A. (1997). *Modelling learning as it happens in a diagramatic reasoning task* (Tech. Report No. 45): ESRC CREDIT, Dept. of Psychology, U. of Nottingham.
Ritter, F. E., & Bibby, P. A. (2008). Modeling how, when, and what learning happens in a diagrammatic reasoning task. *Cognitive Science*.
Ritter, F. E., Haynes, S. R., Cohen, M., Howes, A., John, B., Best, B., Lebiere, C., Jones, R. M., Crossman, J., Lewis, R. L., St. Amant, R., McBride, S. P., Urbas, L., Leuchter, S., & Vera, A. (2006). High-level behavior representation languages revisited. In *Proceedings of ICCM - 2006- Seventh International Conference on Cognitive Modeling,* 404-407. Trieste, Italy: Edizioni Goliardiche.
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., & Schooler, L. J. (2001). The learning curve. In *International encyclopedia of the social and behavioral sciences* (Vol. 13, pp. 8602-8605). Amsterdam: Pergamon.
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.
Siegler, R. S. (1986). *Children's thinking*. Englewood Cliffs, NJ: Prentice-Hall.
Siegler, R. S. (1988). Strategy choice procedures and the development of multiplication skill. *Journal of Experimental Psychology : General, 117*(3), 258-275.
Simon, H. A. (1975). The Functional Equivalence of Problem Solving Skills. *Cognitive Psychology, 7*(2), 88.
Simon, H. A. (1979). *Models of thought*. New Haven, CT: Yale University Press.
Simon, H. A., & Gilmartin, K. J. (1973). A simulation of memory for chess positions. *Cognitive Psychology, 5*, 29-46.
Simon, H. A., & Reed, S. K. (1976). Modeling strategy shifts in a problem-solving task>. *Cognitive Psychology, 8*, 86-97.
St. Amant, R., Freed, A. R., & Ritter, F. E. (2005). Specifying ACT-R models of user interaction with a GOMS language. *Cognitive Systems Research, 6*(1), 71-88.
Staszewski, J. J. (1988). Skilled memory and expert mental calculation. In M. T. H. Chi, R. Glaser & M. J. Farr (Eds.), *The nature of expertise*. Hillsdale, NJ: Erlbaum.
Taatgen, & Anderson, J. R. (2002). Why do children learn to say "Broke"? A model of learning the past tense without feedback. *Cognition, 86*(2), 32.
Taatgen, N. A. (1999). Explicit learning in ACT-R. In U. Schmid, J. Krems & F. Wysotzki (Eds.), *Mind modeling - A cognitive science approach to reasoning, learning and discovery* (pp. 233-252). Lengerich (Germany): Pabst Scientific Publishing.
Taatgen, N. A., & Lee, F. J. (2003). Production compilation: A simple mechanism to model complex skill acquisition. *Human Factors, 45*(1), 61-76.
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.
Yost, G. R. (1993). Acquiring knowledge in Soar. *IEEE Expert, 8*(3), 26-34.
Zimmerman, B. J. (1996). Self-regulated learning of a motoric skill: The role of goal setting an self-monitoring. *Journal of Applied Sport Psychology*, 8.

APPENDIXES

Appendix 1 – Written instructions for the participants

Diagrammatic reasoning study: Doing The Klingon Laser Bank Task.

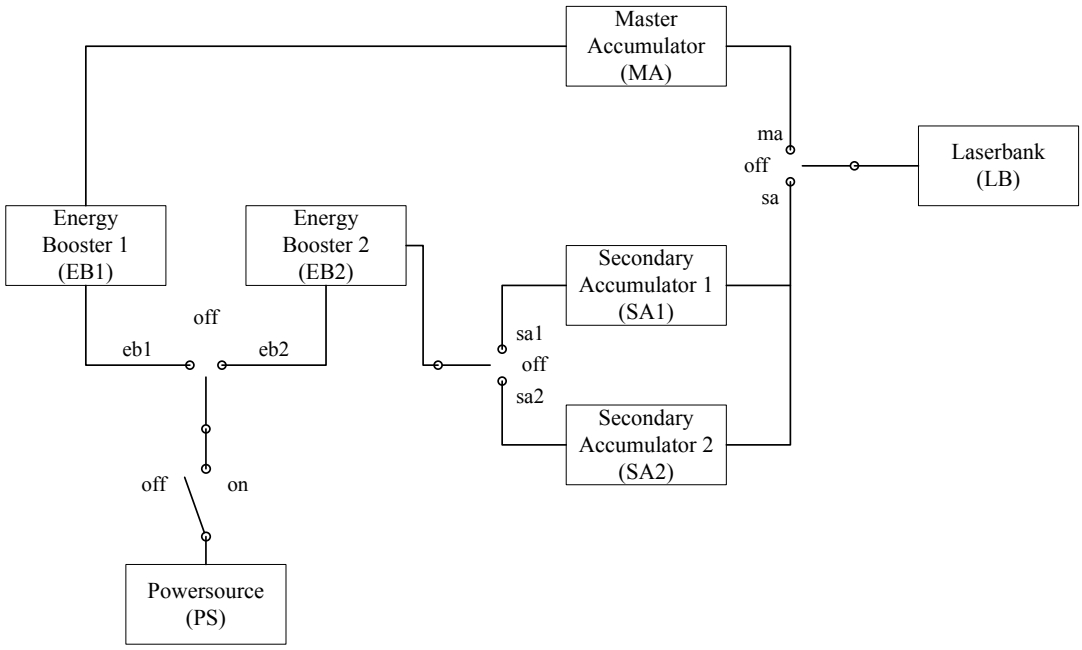
Instructions

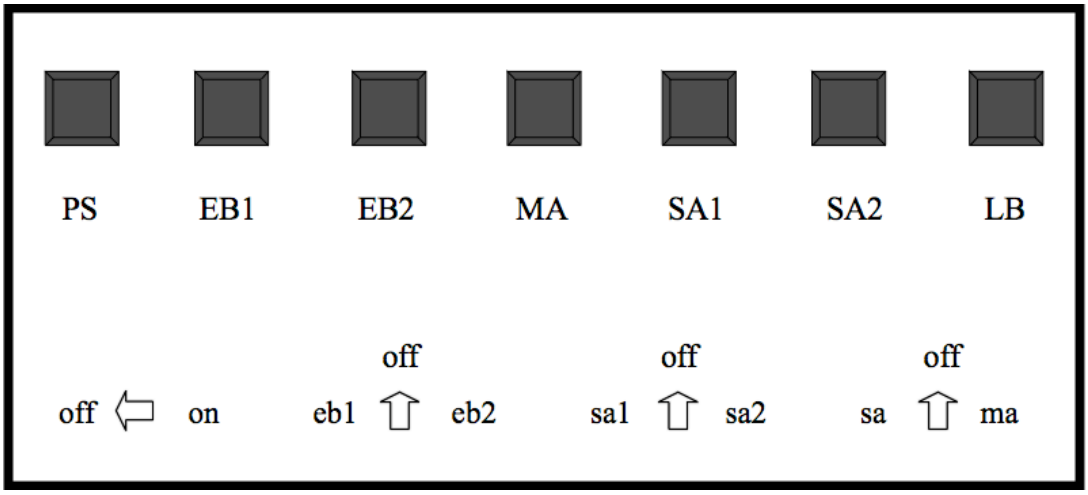
Imagine that you are on board a Klingon warship under attack from the Starship Enterprise for attempting to smuggle arms to the planet Orion III, your new allies. Your job is to operate the new laser weaponry developed using designs based on the phasers on board the Enterprise. The laser system has been 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).

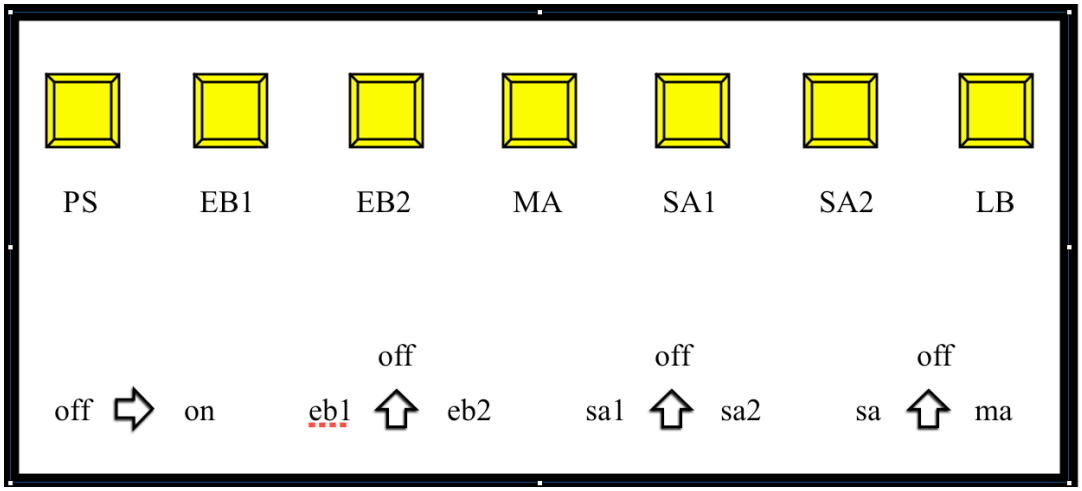
Power is routed through the system by changing the position of switches directing the power from the power source on to one of the energy boosters then to one of the accumulators and finally an accumulator is selected to send power to the Laser Bank.

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





Example of an interface without power



Light working test on the Interface

Frequently Asked Questions List

1. **What is the different between an on or off light?**

 ← This means the component is working.

 ← This means the component has no power or it is broken.

2. **What if the lights or the switches are broken?**

During this task you can assume that everything works except for the one component that is broken.

3. **What if two components are broken?**

In every case, there is only one broken element.

4. **Where do I have to click while doing the task?**

You always click the light of the part that you think is broken.

5. **Do I get to know if I'm correct or incorrect?**

No, you do not get any direct feedback; we will show you the correct answers after the task.

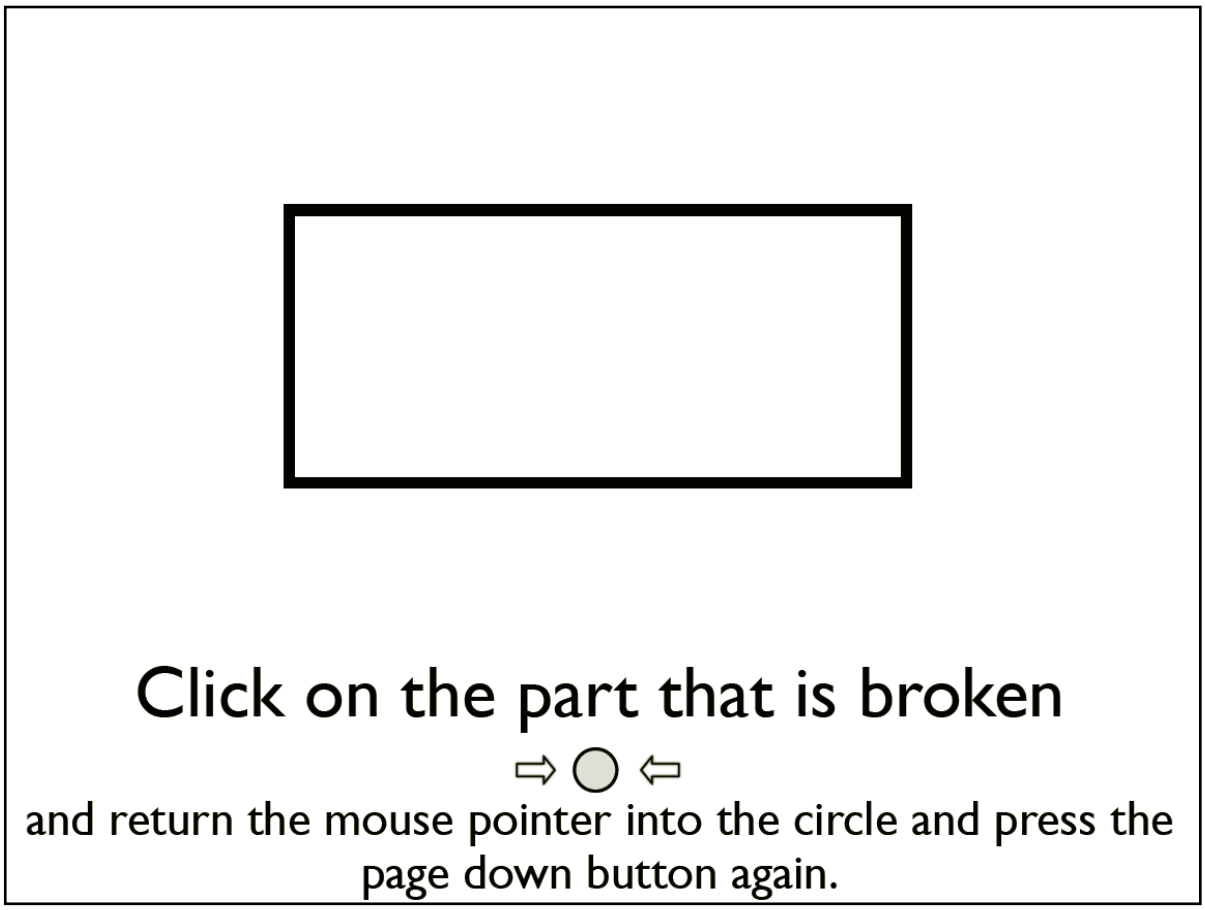
6. **What happens if I click on the wrong part?**

Nothing, we will analyze your choices later and no one will see your individual performance.

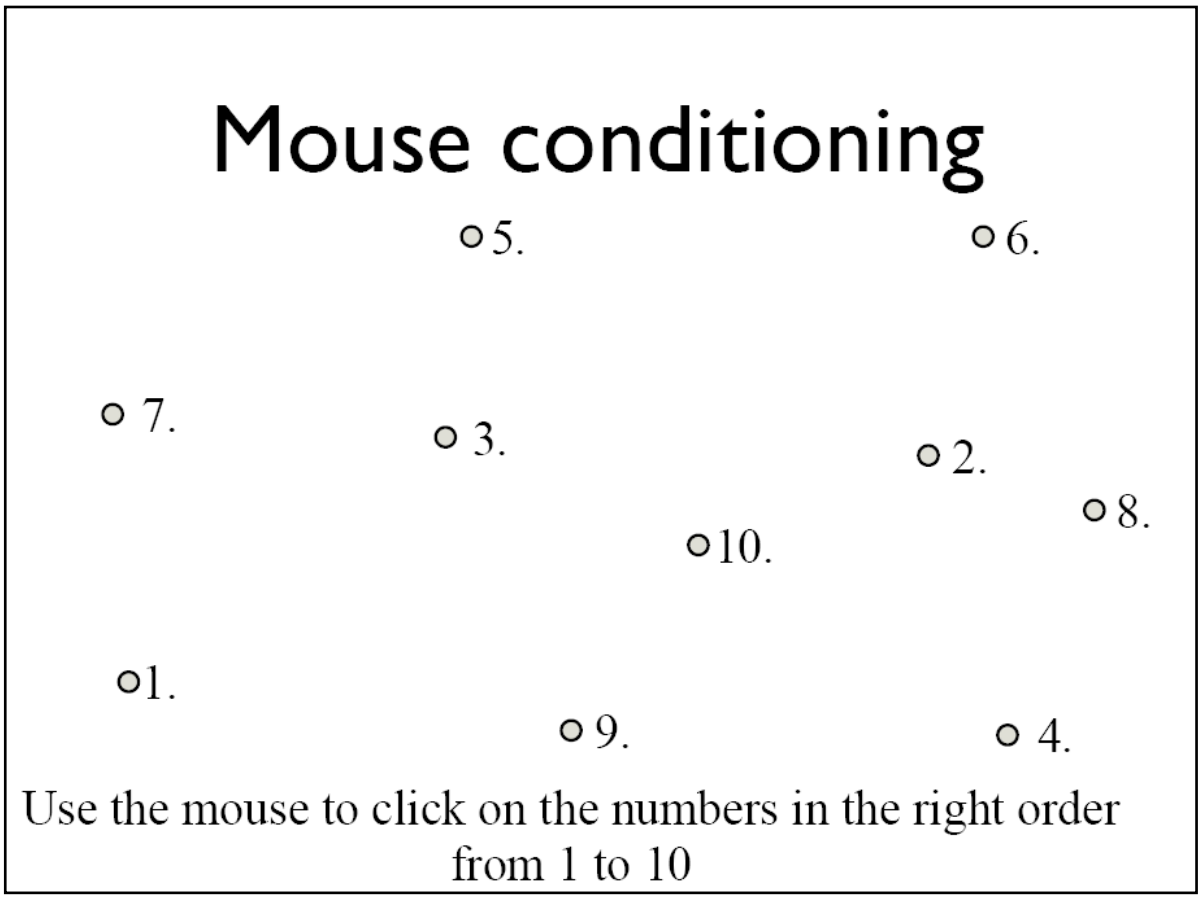
7. **Do I have to guess to get the correct answer?**

No, there is always a way to see which part is broken.

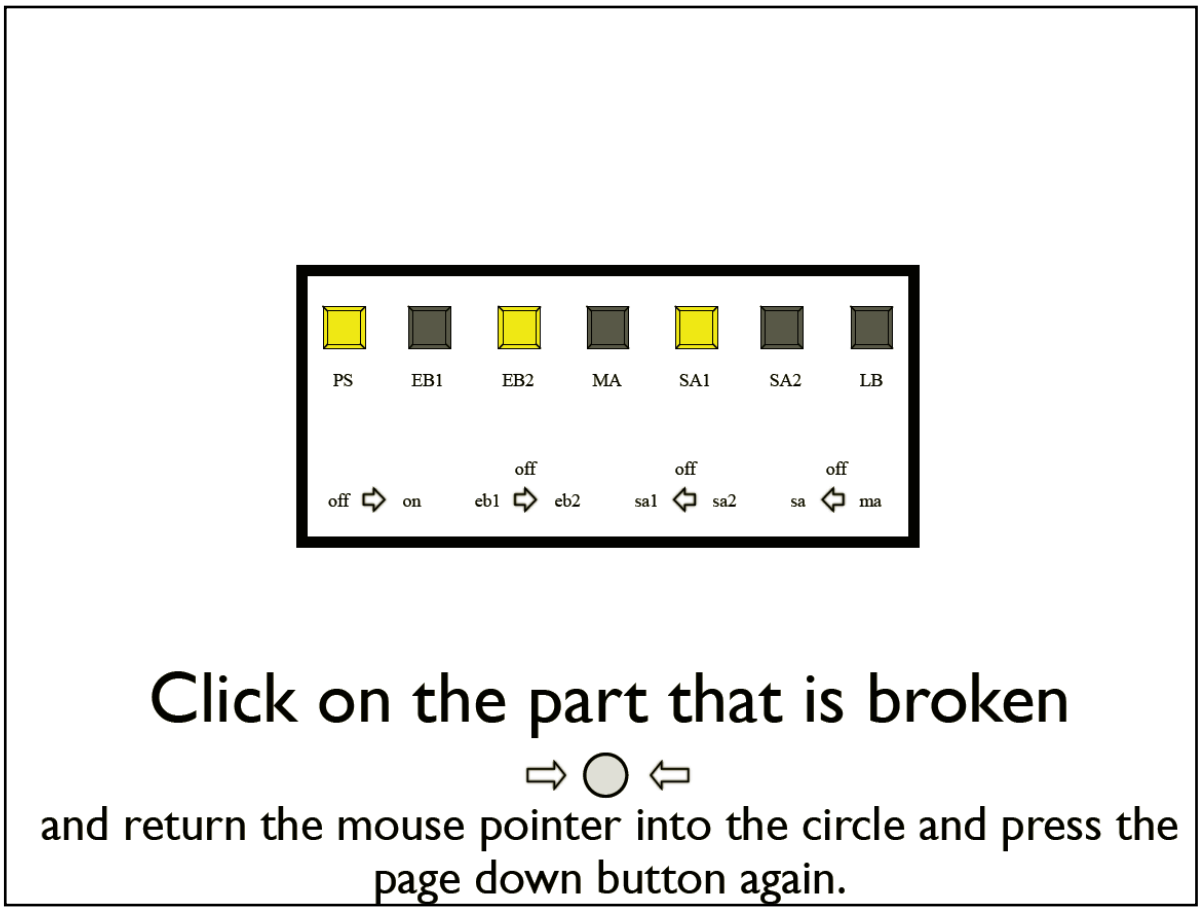
Appendix 2 – Task environment introduction



Appendix 3 – Mouse conditioning



Appendix 4 – Task environment example



Appendix 5 – Participants and their majors

Table 18. All participants, their study major, and MacMouse experience.

Participant	Study major	MacMouse experience
D3_9	business informatics	no
D7_3	business informatics	no
D8_4	business informatics	no
D9_5	business informatics	no
D16_11	business informatics	no
D18_11	business informatics	no
D19_11	business informatics	no
D20_11	business informatics	no
D2_8	computer science	no
D4_10	computer science	no
D6_2	computer science	no
D10_6	computer science	no
D12_8	computer science	no
D13_9	computer science	no
D14_10	computer science	no
D15_11	computer science	no
D17_11	computer science	no
S4_1	computer science	yes
S10_6	computer science	no
S13_3	computer science	yes
D1_7	economics	no
D5_1	economics	no
D11_7	economics	no
S2_9	economics	no
S7_5	economics	yes
D21_11	mechanical engineering	no
S1_3	mechanical engineering	yes
S5_4	mechanical engineering	yes
S9_7	mechanical engineering	yes
S3_8	psychology	no
S6_2	psychology	no
S8_10	psychology	yes
S11_1	psychology	no
S12_2	psychology	yes
S14_4	psychology	no
S15_5	psychology	no
S16_6	psychology	yes

Appendix 6 – Participants and their two best strategies

Table 19. Participant, their strategy, correlation (R), intercept (in seconds), regression coefficient (B, in seconds per model cycle) when participants are regressed to their two best fitting strategies.

Participant	Strategy	R	Intercept	B
D1_7	Diag	0.51	5.41	0.04
	DiagSelect	0.39	2.63	0.179
D2_8	CheckLitSwitch	0.51	4.15	0.127
	Diag	0.49	5.79	0.02
D4_10	PrevState	0.451	4.861	0.094
	CheckLitSwitch	0.185	10.085	0.085
D6_2	DiagSelect	0.409	4.921	0.109
	PrevState	0.334	12.602	-0.025
D7_3	PrevState	0.372	7.697	-0.020
	DiagSelect	0.337	4.026	0.038
D9_5	DiagSelect	0.53	5.64	0.100
	CheckLitSwitch	0.52	6.56	0.096
D10_6	DiagSelect	0.363	3.937	0.197
	CheckLitSwitch	0.304	7.276	0.152
D11_7	DiagSelect	0.380	5.548	0.169
	PrevState	0.227	15.527	-0.029
D12_8	Diag	0.64	3.89	0.01
	LitIndicator	0.46	3.85	0.024
D13_9	PrevState	0.61	1.63	0.055
	CheckLitSwitch	0.35	3.91	0.107
D15_11	DiagSelect	0.56	0.56	0.155
	CheckLitSwitch	0.41	3.17	0.110
D16_11	LitIndicator	0.438	9.140	0.110
	Diag	0.31	12.83	0.018
D18_11	DiagSelect	0.53	4.34	0.065
	PrevState	0.43	9.06	-0.018
D19_11	PrevState	0.62	12.05	-0.037
	DiagSelect	0.47	4.57	0.078
D20_11	LitIndicator	0.375	5.282	0.107
	Diag	0.22	9.37	0.013
D21_11	DiagSelect	0.338	5.125	0.075
	CheckLitSwitch	0.196	6.773	0.043
S1_3	CheckLitSwitch	0.57	4.42	0.253
	DiagSelect	0.53	3.12	0.238
S3_8	LitIndicator	0.258	7.787	0.042
	CheckLitSwitch	0.186	8.402	0.046
S6_2	LitIndicator	0.306	14.811	-0.065
	PrevState	0.279	14.118	-0.030
S8_10	LitIndicator	0.347	6.148	0.101
	PrevState	0.330	6.636	0.057
S10_6	LitIndicator	0.55	-1.18	0.163
	DiagSelect	0.35	2.48	0.103
S11_1	DiagSelect	0.349	5.678	0.116
	PrevState	0.289	7.662	0.033

S13_3	DiagSelect	0.475	2.430	0.130
	CheckLitSwitch	0.432	3.954	0.120
S14_4	DiagSelect	0.356	6.205	0.086
	Diag	0.29	8.39	0.012
S15_5	DiagSelect	0.68	2.18	0.109
	CheckLitSwitch	0.62	3.53	0.098
S16_6	CheckLitSwitch	0.450	9.127	0.121
	Diag	0.43	11.02	0.018

Eidesstattliche Erklärung

“Ich erkläre hiermit gemäß § 17 Abs. 2 APO, dass ich die vorstehende Masterarbeit selbständig verfasst und keine anderen als die angegebene Quellen und Hilfsmittel benutzt habe.”

30. April 2008
(Datum)

_____
(Unterschrift)