Modeling a Strategy with Learning in a Complex Task

Shan N. Wang (sxw820@psu.edu)
Frank E. Ritter (fer2@psu.edu)
College of IST, Penn Sstate
University Park, PA 16802 USA

Keywords: individual differences; cognitive modeling

Introduction

We used a complex electrical troubleshooting task shown in Figure 1 to study problem solving with learning, the BenFranklin Radar System that consists of 36 components, versus 7 components in the Klingon Laser Bank task used in several previous studies (Friedrich & Ritter, 2020; Ritter & Bibby, 2021). MENDS is a simulator created by Charles River Analytics for the BenFranklin Radar system, shown in the lower figure, used under license. The participants' task is to find the broken component in the circuit. In MENDS, participants can click and open the subsystem (trays) to see the components in each subsystem. They can decide and click the component that they think is the broken component, based on their schematic knowledge, the light and switch conditions. Components without power are in grey.

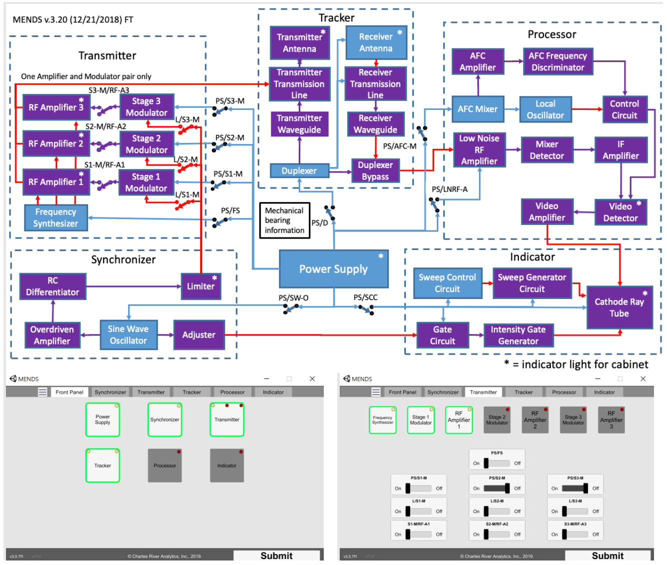

Figure 1: Upper: Schematic for the BenFranklin Radar. Blue lines are power; red lines are signal; purple lines are both. Lower: the MENDS simulator's front panel and one subsystem.

We collected participants' mouse moves and clicks for our modeling and data analysis using the RUI logger (Kukreja et al., 2006). To gather data, a user study was run (Ritter et al., 2022). The goal was to identify strategies and learning with a larger number of participants. After data cleaning, we had 111 participants' data in the test session, where they were asked to finish 20 problems. We collected the component and times the participants clicked on. From the mouse clicks of the top 6 participants in the test session, we developed and implemented four strategies in the BenFranklin Radar task. Here we present one, the Grey Upstream strategy. The observed time for participants' performance was compared with the predicted time of our strategy models, without and with learning.

Modeling

Figure 2 shows how we built a simple task model for MENDS in Python, also described in (Ritter et al., 2022). The simple task model used a Panda data frame to store and reflect the components' broken status (1: component is fine; 0: broken), light status (1: component has light on; 0: light off), downstream components to give power, upstream components to receive power, switch condition before them (1: switch on; 2: switch off), the number of times the required schematic knowledge have applied by participants.

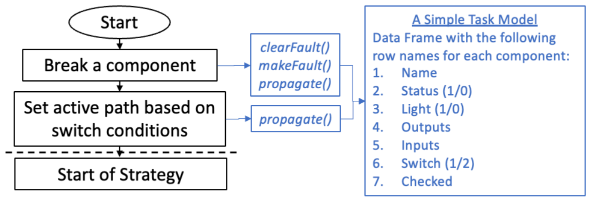

Figure 2. A flowchart for the simple task model and the grey-upstream strategy model. An active path refers to the components receiving power and that are supposed to have lights on.

The Grey Upstream Strategy

The Grey Upstream strategy is one of the four strategies that may be used to identify the broken component. Here we define the strategy in the scope of a task. Those strategies are categorized by four features: their starting point, how the front panel information was used, degree of schematic knowledge used, and degree of interface information used. Variations within one strategy are also possible. All strategies find the correct fault. The grey upstream strategy (GreyUp), shown in Figure 3, is based on participants P324, 420, 451, & 453. The strategy involves two major steps, finding a grey component as a starting point and tracing upstream in the schematic till finding the broken component. To locate the starting point, users click into the first grey tray using light information from the front panel and identify the first grey component by interface order from left to right and up to down within the clicked tray. Starting from the first grey component, participants use their schematic knowledge to trace up until they identify the broken component, which is the only one with its light off but all its upstream components in the active path are with their lights on.

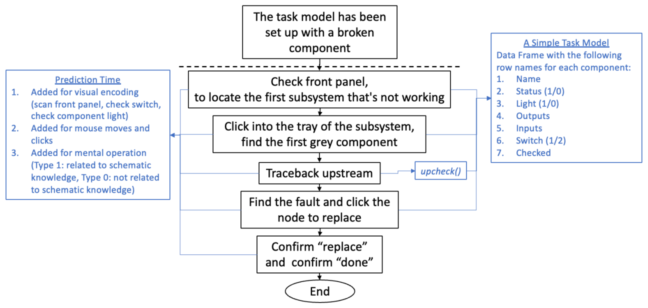
The task model has been set up with a broken component
Check front panel, to locate the first subsystem that's not working
Click into the tray of the subsystem, find the first grey component
Traceback upstream
upcheck()
Find the fault and click the node to replace
Confirm "replace" and confirm "done"
End
Prediction Time
1. Added for visual encoding (scan front panel, check switch, check component light)
2. Added for mouse moves and clicks
3. Added for mental operation (Type 1: related to schematic knowledge, Type 0: not related to schematic knowledge)
A Simple Task Model
Data Frame with the following row names for each component:
1. Name
2. Status (1/0)
3. Light (1/0)
4. Outputs
5. Inputs
6. Switch (1/2)
7. Checked

Figure 3. A flowchart of the Grey Upstream strategy, as a technical detail example.

The models take steps to solve the task based on each strategy, and time is added with each type of action. The time for each step depends on the learning level. Under the learning condition, we assume that knowledge is transferred to later tasks and the participants solve the later tasks faster based on the repetitive application of the same knowledge. Time parameters used in the models include visual coding, 0.4 (s); mouse move, 1.1; mouse click, 0.2; mental operation, 1.35; learning rate, 0.4. The learning follows ACT-R and the power law, and the base times are KLM times.

Models learn by modifying the mental operation time. We do this by having mental operation time as a function and not a constant. The function *mental(type, var2)* gives different times for different types of mental operation and learning levels. The *type* variable includes *type 1* and *type 0*. *Type 1* refers to the mental operation time retrieving a component by using schematic knowledge. *Type 0* refers to any other mental processing not related to schematic knowledge. *Type 1*, retrieving a component, can be faster with learnings which indicated by *var2*. For *type 0*, *var2* is a random number because the time is assumed to be fixed, and no learning happens. For situations that assume no learning, the mental operation function in models uses a constant 1.35 s. When learning, the mental operation time is $\mathbf{1.35 \cdot n^{-l}}$, **where** n is the number of times the component has been checked or the number of times the required circuit knowledge has being used. The value of *l* represents the learning rate, 0.4.

Comparison

We trained the models with the previous sessions that participants experienced before we ran the models for tasks in the test session. We compared the predicted time to the observed times of the participants. We consider that participants fit well to a strategy if they have R^2 with a p-value < .05. 14 participants' behaviors match a strategy ($p < .05$; R^2 varied) without learning in the test session. 69 participants' behaviors match a strategy with learning in the test session.

Figure 4 shows the match of PID 413 as an example. PID 413 has R^2 of .498, without learning and an R^2 of .518, with learning. The red dotted line is the observed time from human data; the blue solid line is the predicted time without learning; the black solid line is predicted time with learning and was trained with previous sessions' faults.

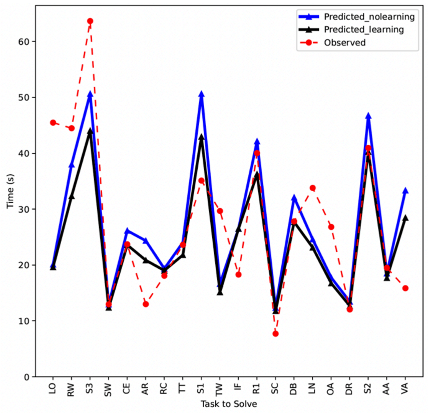

Figure 4: An example of PID 413 as one of the best matches. A comparison of human data, our Grey Upstream Strategy model with learning, and without learning.

Discussion and Conclusion

The strategy that includes learning indeed does better at performance prediction. Also, more strategy models could have been presented. Variations within strategies and strategy switch are not yet modeled. The current strategy models are from the top 6 well-performed participants, while the other participants made much more errors during the fault-finding process. We have not modeled errors, lapses, or changes of strategies within the same session. Modeling those can be our future steps. If our strategy models consider errors, the match between participants and strategies may increase.

Acknowledgements

Thanks to Sarah Ricupero, Mizzah Tocmo, and Rochelle Lorraine Clerkin for providing feedback to the longer version of the paper.

References

Friedrich, M. B., & Ritter, F. E. (2020). Understanding strategy differences in a fault-finding task. *Cognitive Systems Research*, *59*, 133–150.

Kukreja, U., Stevenson, W. E., & Ritter, F. E. (2006). RUI: Recording user input from interfaces under Windows and Mac OS X. *Behavior Research Methods*, *38*(4), 656–659.

Ritter, F. E., & Bibby, P. (2021). Modeling how and when learning happens in a simple fault-finding task. *Proceedings of the 2001 Fourth International Conference on Cognitive Modeling*, 330–341.

Ritter, F. E., Workman, D., & Wang, S. (2022). Predicting learning in a troubleshooting task using a cognitive architecture-based task analysis. *International Conference on Cognitive Modeling. 222-223.*

.