Description of the new Utility Learning mechanism for ACT-R 6 by John Anderson

I am describing below two sets of ideas about the utility of production rules in ACT-R. The first set of ideas constitutes a simplification of the current system in ACT-R that is quite complex with separate tracking of probability of success and cost. There are also separate rules for adjusting utility of new created rules versus established rules. I will devote most of this note to that. Then I spend a little time discussing ideas to complicate matters to deal with issues of exploration-exploitation.

The Simplifying Idea

The proposal is to use the simple linear model of Bush and Mosteller. If U_i(n-1) is the utility of a rule i after its n-1st application and R_i (n) is the reward the rule receives for its nth application, then its utility U_i(n) after its nth application will be

$$U_i(n) = U_i(n-1) + \alpha[R_i(n) - U_i(n-1)] \qquad \textbf{Difference Learning Equation}$$

where α is the learning rate and is typically set at 0.2. This is also basically the Rescorla-Wagner learning rule (Rescorla & Wagner, 1972) and matches the evidence about reinforcement learning. It gradually adjusts the utility of a rule until that utility is the average reward.

There are a couple of interesting issues about the reward itself. Rewards occur at various times not exactly associated with any production rule – for instance, in the case of a monkey it may press a button and receive a squirt of juice a second later. A number of rules may have fired before the reward is delivered. The reward that production i will receive will be the external reward received minus the time from the rule's selection to the reward. This serves to give less reward to more distant rules. This is like the temporal discounting in reinforcement learning but proves to be more robust within the ACT-R framework (not suggesting it is generally more robust). In ACT-R models it is necessary to place in event markers so that this mechanism knows how far to go back in time in crediting each reward. A new reward will cause the utility of all productions that have fired since the last reward to be updated.

Another complication concerns what to do when a new rule is created. New rules are created by collapsing two rules (parent rules) into a single rule that does the work of both. The proposal is to give the new rule an initial strength of 0 but each time it is recreated its strength is increased according to the Difference Learning Equation. The reward attributed to the rule is the utility of the first parent rule.

With one more complication we have a mechanism that can learn to make choices among rules. This is that we assume some momentary noise in the utility values. If there is a set of productions that can apply, then the probability of any one production applying will be the probability that it has momentarily the highest utility. If U_i is the expected utility of the rule determined by the difference learning equation above and U_j is the expected utility of the competing productions j, then the probability that production i will be selected is given by:

$$P_i = \frac{e^{U_i / s}}{\sum_j e^{U_j / s}}$$

Conflict-Resolution Equation

In this equation s is a parameter that controls the noise in the utilities, and we conventionally set this to 1. This choice rule is a very common rule that goes back to Luce (ref), serves as the selection mechanism in Boltzman machines (where s is temperature), and has a wide variety of other applications in cognitive models, particularly those of a connectionist persuasion.

To illustrate how these utility calculations work, consider how the following four rules might fare:

Retrieve-Instruction (Reward 10)
If the goal is to process a column
Then retrieve an instruction for that kind of column

Request-Difference-Subtract (Reward 14)
If the goal is to process a column
   and the top digit is larger than the bottom digit,
Then subtract the bottom from the top

Request-Difference-Borrow (Reward 14)
If the goal is to process a column
   and the top digit smaller than the bottom digit,
Then subtract the top from the bottom

Request-Difference-Wrong (Reward 14 or 0)
If the goal is to process a column
Then subtract the larger from the smaller

The first rule retrieves instructions appropriate for processing a column. The second and third rules reflect new alternatives that might be compiled from using correct instructions. The reason why the second and third have higher rewards is that they produce the result (a correct answer in the column – which we will assume is reinforcing) without the effort of having to retrieve the original instructions and interpret them. The last rule represents the most common bug in children's subtraction. It has been argued that this bug originates essentially from incorrect self-instruction (VanLehn & Brown, ref; VanLehn, 1990). It sometimes works in which case it will be every bit as rewarding as the correct rule. When it leads to a wrong result we assume that there is no reward. We ran the ACT-R model with standard parameters through 100 randomly generated subtraction experiences. The first rule, Retrieve-Instruction, started with its utility already at 10 and that the other rules began with a utility of 0. Initially, the model will exclusively use the retrieve rule and apply the instruction. Just for illustration purposes, I assumed that the student is just as likely to create the correct rule (**Request-Difference-Subtract** or **Request-Difference-Borrow)** as the incorrect interpretation (**Request-Difference-Wrong)** each time **Retrieve-Instruction** applies.

Figure 1 shows the growth utility (averaged over 1000s of runs) over the 100 trials (with $\alpha = 0.2$). The rule that retrieves instruction holds steady at 10 while the other three grow initially collecting the reward of the correct instruction rule (their parent). The wrong rule grows fastest because it applies in all the situations, whereas the correct rules apply to only a subset of the situations. The borrow rule grows slowest because it is required in the fewest cases. However, eventually the rules reach a range (about 8) where they are sometimes selected rather than the instruction rule. This leads to a growth in actual experience and the wrong rule suffers a slight depression while the other two rules move past the instruction rule to their eventual steady-state utilities of 14. Figure 1b shows the probability that a rule will be selected when applicable (the instruction rule and wrong rule are always applicable; the other two are mutually exclusive). The probabilities in part b can be calculated from the utilities in part a using the conflict resolution equation above with a value of 1 for s.

It is worth emphasizing the set of features of this treatment of utility:

1. **Gradual Introduction of a Rule:** Initially when a new rule is created it has 0 utility and it will not be tried. Rather, the system will continue to use the pair of parent rules that gave rise to it. However, every time it is recreated its utility will be increased until it is occasionally tried rather than its parent. For purposes of the utility learning it will be given the reward of the parents. Gradually, its strength will increase until it reaches a point where it replaces its parents. That completed rule would lead to the same results as the original pair of rules but eliminates a production firing and a costly declarative retrieval. This is basically the advantage of the subtract and borrow rules in Figure 1.
2. **Ordering of Rules by Utility:** If there are different rules that lead to different outcomes it will learn to choose the rule with the highest outcome. For instance, in Figure 1, faced with a buggy subtraction rule that sometimes leads to errors and a correct rule, the model came to prefer the correct rule – that is, based on a higher utility being assigned to getting a problem correct than just solving it.
3. **Sensitivity to Solution Time:** Faced with two rules that produce the same result but with one faster than another, the model will come to prefer the faster because the utility of a rule is defined as the difference between the reinforcement and the delay. This is the basis for the preference for the subtract and borrow rules over the rule that just retrieved the instruction. This requires that reinforcement and time be placed on the same scale, and in the ACT-R models that scale is time in seconds, but this is basically just a technical matter.
4. **Sensitivity to Change:** The system is not locked into one way of evaluating rules and should the payoff for rules change it can adjust. This does not seem to happen in a domain like subtraction where the rules never change, but we will

consider such a situation in the next section where people had to adapt to changing probabilities.

Probability Learning

Back in the early days of mathematical psychology, Friedman et al completed an experiment on probability learning that is a marvel in detail of reporting. This task proves to a perfect case for illustrating the details of utility learning. The task is an experimental version of guessing whether a possibly biased coin will come up heads or tails. Participants were to guess which of the two outcome lights would come on. Task instructions encouraged participants to try to guess the correct outcome for each trial. The study changed the two buttons' success probabilities for each 48-trial block in the experiment. Specifically, for the odd-numbered blocks 1-17, the probabilities of success of the buttons (p and 1-p) are 0.5 where p designates the right button. For the even-numbered blocks 2-16, p took on the values from 0.1, to 0.9 in a random order. Therefore, in the first block, the two buttons were equally likely to be correct. Starting from the second block and in each of the subsequent even-numbered blocks, one of the buttons was more likely to be correct.

Friedman et describe the exact sequence of trials subjects saw in each of the odd number blocks. There were 48 trials in each block, and Friedman et al. provided the mean choice proportions of each button in every 12-trial sub-block. While they used many different orders they kept track the odd-block that an even .5 block followed and classified the performance in each .5 block accordingly – so one can track the effect of prior experience on these .5 blocks. Figure 2 shows the correspondence between the data and the predictions of the utility learning mechanism we described. The data reported are the numbers of choices of the right button out of 12 opportunities. The data are presented as if the odd blocks systematically began with 0.1 and incremented to 0.9. None of the 10 sequences used by Friedman et al did this but we have sewn the data back together as if this were the case. The .5 block after each odd block is based on the average numbers they report for the .5 block that followed that probability.

The ACT-R model assumed that the critical choice was between two rules for choosing buttons. The only parameter estimated in fitting the 68 data points in the figure was the reward for correctly guessing which was set at 2.5 units. The fit is really quite remarkable. For instance, notice in the .8 block that the number of right button choices reaches a maximum in the second sub-block and drops off a little. This is because the actual sequences (there were two) used for the .8 blocks had, by chance, a larger number of left choices correct in the second half.

Friedman et al also report a wide variety of statistics on sequential effects. For this purpose they ran 6 48-trial blocks after the set represented in Figure 2 to get data on sequential changes when the overall probability was stable. The probability of the right response was .8 during these trials. One statistic they looked at was how the length of run during this time affected the probability of a right response. This is plotted in Figure 3 where a negative number reflects a run of lefts and a positive number a run of rights. The data is plotted for all cases where the total number of observations is greater than 100. There are many more cases of runs of rights than runs of lefts and so the plot is from -3 (3 lefts in a row) to 18 (18 rights in a row). The predictions of the theory are also given in the figure. Again the correspondence is compelling. One thing this plot makes clear is that, unlike some probability learning experiments, participants were not displaying the gambler's fallacy and guessing the opposite after a long run. Friedman et al took some effort to avoid this and so give us a relatively pure test of utility learning where there is not interference of higher-order hypothesis testing.

Figure 4 displays other measures of sequential statistics. Here we are looking at the choice a subject makes on the current trial C_0 as a function of their choice and the feedback on the prior trial (C_{-1} and F_{-1}) and the trial one prior to that (C_{-2} and F_{-2}). Thus, for instance LRRRL stands for the event where they chose left two trials back and got right as feedback, chose right one trial back and got right as feedback, and now chose left. Figure 4a displays the proportions of these 32 trial types. As can be seen the correspondence is pretty good. Figure 4b collapses over the prior choices of the participant and just plots proportion of right choice as a function of the prior two outcomes. As this displays, when one of the two previous answer's outcomes was right and the other was left they are more likely to choose right when the more recent outcome was right. This recency effect is part of the nature of the utility updating – the most recent event has the strongest effect.

Exploration-Exploitation

One of the curious features of the probability-learning paradigm is the complimentary nature of the choices. If one action is not tried the other must; if one action has low reward the other has high. If one button starts to get low utilities the other will be tried and the participant will begin to be rewarded for it. This makes it relatively easy for a participant to follow the switches in the task. This is not true for many situations where just because one action is no longer profitable, it does not mean that some other action has to be performed or that it will be successful. In foraging, for instance, just because one patch no longer has food there is no reason to suspect that another patch will suddenly become more profitable. Nonetheless, it is important to begin to explore other options at these critical junctures.

Figure 5 from Aston-Jones et al (1997) illustrates such a situation. Monkeys had been highly overtrained to respond to a sequence of vertical and horizontal bars. One sort of bars occurred 80% of the time and responding to them with a lever press only brought a 3 second time out. The other sort of bar occurred only 20% of the time but responding to them with a lever press obtained a drop of juice. Then everything reverses – the relative frequency of the two bars changes and the other (now low-frequency) bar was reinforced. This happens at about time 17 min in Figure 5. As can be seen, before the switch, the monkey had things down nearly perfectly, responding to almost all of the bars that paid off (hits) and responding to almost none of the bars that did not (false alarms). The data in Figure 5 was for one specific case, but the monkeys averaged over 99% hits and less than 1% false alarms. After the switch there is a tendency to continue to respond to the former reinforced bars (now false alarms) and responses grow to the former non-reinforced bars (now hits). Over the next 500 trials after the switch, the monkeys average

88% hits but also average 29% false alarms. Gradually, they reach the performance before the switch and then the bars are switched again. There were 3 or 4 such reversals -- which is apparently less than is needed for any learning set to begin to show up.

After searching parameters, the best-fitting values that we were able to obtained produced 98% hits and 2% false alarms in the asymptotic stage before the switch – which is nearly as good as the monkeys. After the switch, the model also matched on the 29% false alarms. However, it was only able to generate 50% hits rather than the observed 88%. By decreasing the negative penalty associated with the time-out we could get the hit rate after the switch up to 88% but at the cost of raising the false alarm rate before the switch to 8% and the false alarm rate after the switch to 39%. Basically, it seems impossible to get the high performance before the switch where the animal almost never responds to the bar and have it quickly switch to responding to the bar after the switch. Since it almost never responds to the bar before the switch, it cannot quickly discover that it is now rewarded.

We have been experimenting with an idea to deal with this situation. The idea is to increase the noise in the utility assessments after the switch basically because the animal is getting "frustrated" with the lack of success. However, it is an interesting matter how to actually operationalize the detection of lack of success. It is hard to propose anything with much confidence, but I will describe the current idea.

We assume that the system maintains a global long-term estimate and a global short-term estimate of the rate of rewards and compares the two. These are updated by the same learning rules as are used to update the utilities of particular productions:

$$SR(n) = SR(n-1) + \alpha_{sr}[R(n) - SR(n-1)]$$
$$LR(n) = LR(n-1) + \alpha_{lr}[R(n) - LR(n-1)]$$

where SR is the short-term global estimate and LR is the long-term global estimate. Unlike the specific production utilities, these rates are updated no matter what production fires. The difference between these two are the learning rates α_{sr} and α_{lr} where we set α_{sr} =0.008 and α_{lr} = 0.001 which means the short-term rate is updated 8 times as fast as the long-term rate.

Then we adjust the noise parameter s according to another linear equation:

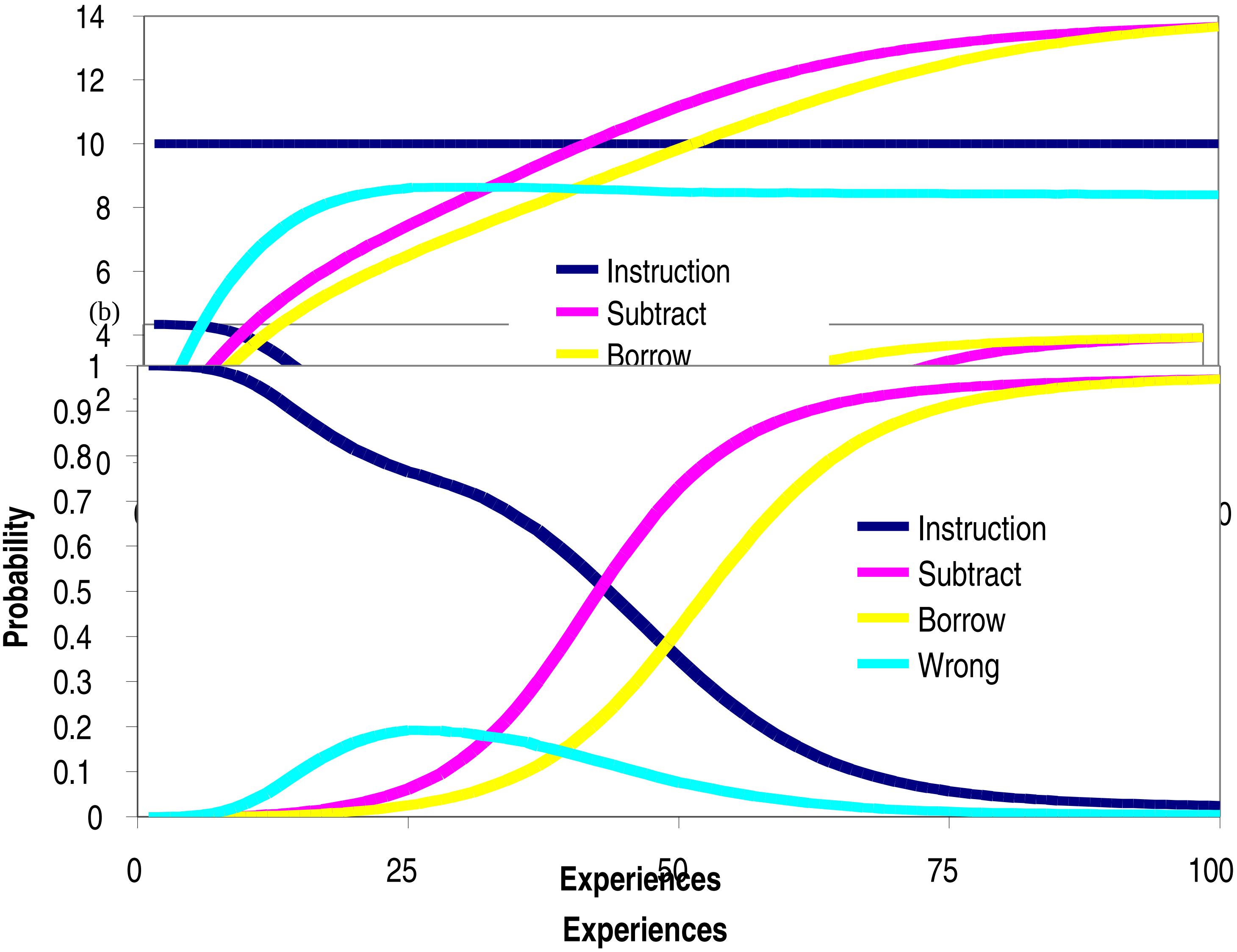
$$s(n) = s(n-1) + \alpha_s[F(n) - s(n-1)]$$
$$F(n) = \max(1, b * (LR(n) - SR(n)))$$

This model is tolerant of some discrepancy between the two rates but whenever that discrepancy is greater than 1/b it will tend to drive the noise rate up. The noise rate will never drop below the base of 1. This involves yet another two parameters α_s which we set to .05, and the scale b, which we set to 4. In addition to these parameters there is the standard utility learning rate α, which we set to .07; the value of the juice, which we set to 50; and the negative value of the time-out, which we set to -8. With these parameters all set we were able to match the high level of performance in the stable period before the switch, keep the false alarm rate after the switch at 28%, and still get 86% hits.

These are a good many additional parameters and equations to have to introduce to basically get one data point to match – the hit rate after the switch. This one experiment clearly does not justify the complications. The question is whether these complications might be justified on some broader view of issues of exploration versus exploitation.

Figure 1

(a)



14
12
10
8
6
4
2
0
Instruction
Subtract
Borrow
Experiences
Probability
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0
25
50
75
100
Instruction
Subtract
Borrow
Wrong
Experiences

(b)

Figure 2

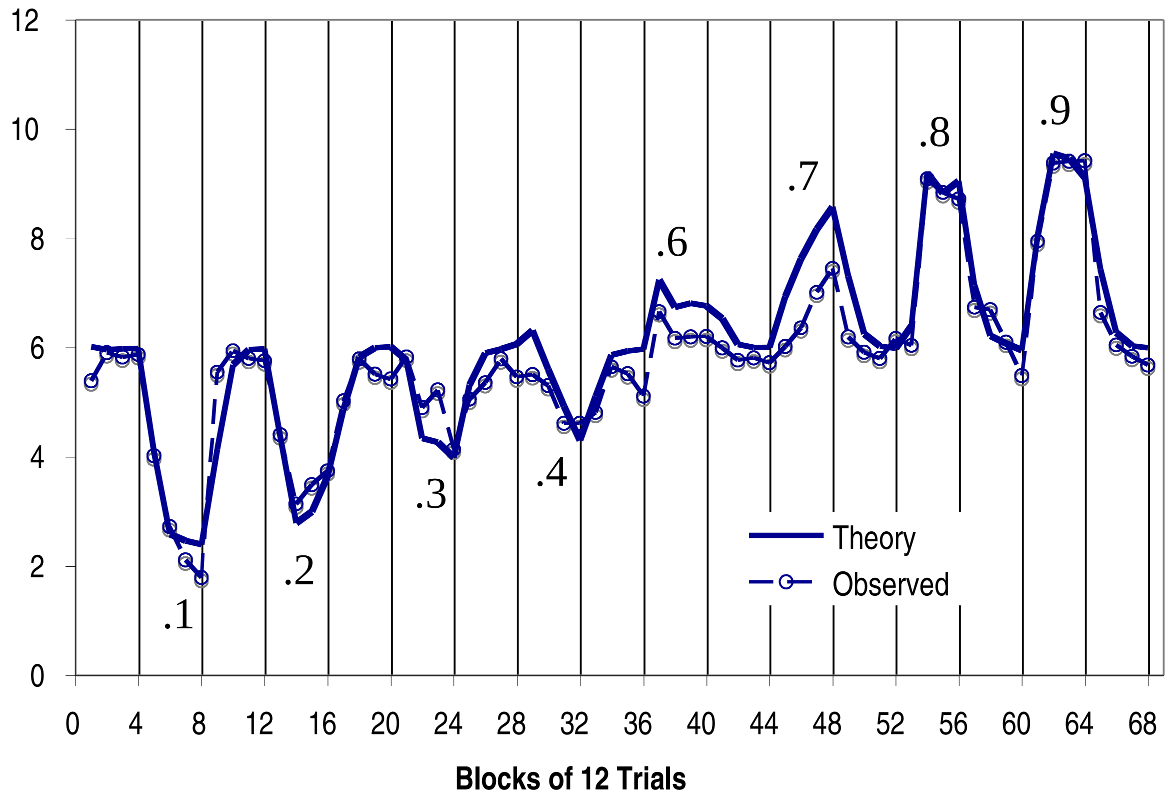

12
10
8
6
4
2
0
0
4
8
12
16
20
24
28
32
36
40
44
48
52
56
60
64
68
Blocks of 12 Trials
.1
.2
.3
.4
.6
.7
.8
.9
Theory
Observed

Figure 3

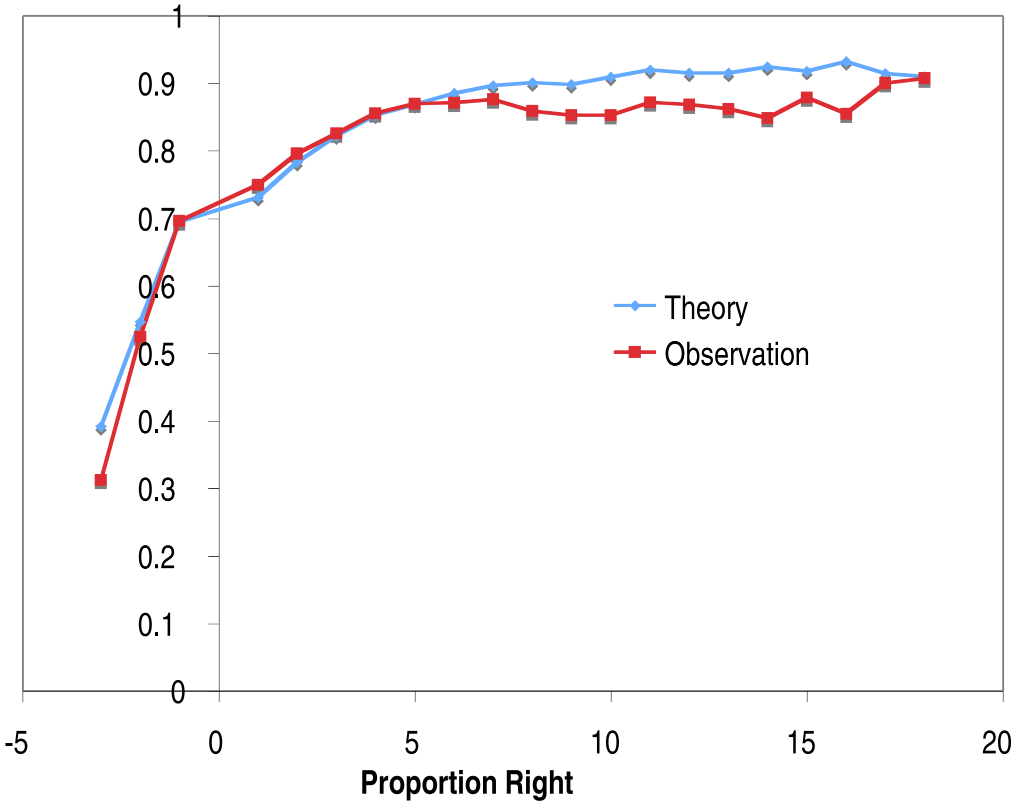

1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
-5
0
5
10
15
20
Proportion Right
Theory
Observation

Figure 4

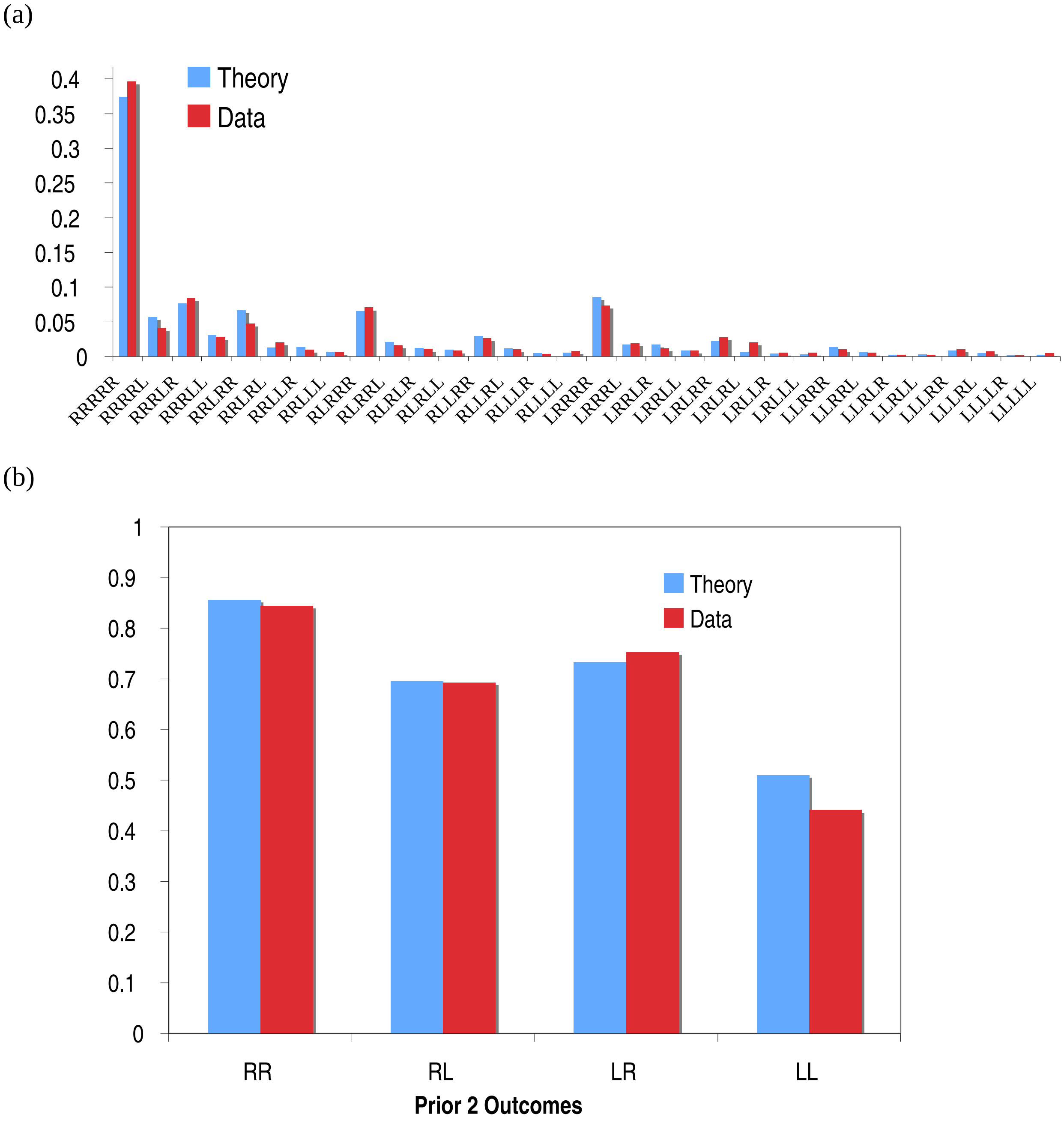
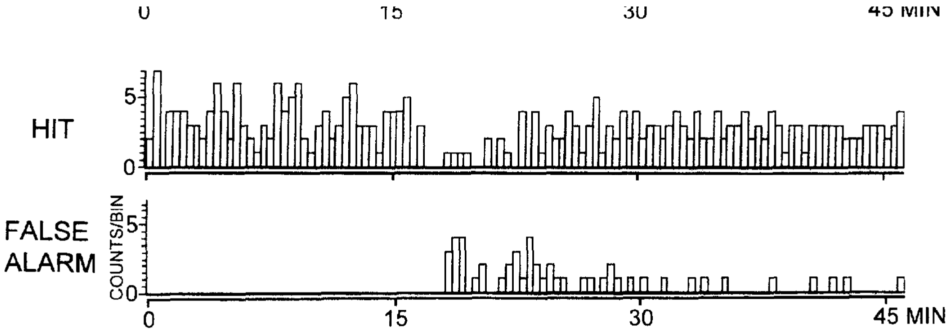

(a)
0.4
0.35
0.3
0.25
0.2
0.15
0.1
0.05
0
🟦 Theory
🟥 Data
RRRRR
RRRRL
RRRLR
RRRLL
RRLRR
RRLRL
RRLLR
RRLLL
RLRRR
RLRRL
RLRLR
RLRLL
RLLRR
RLLRL
RLLLR
RLLLL
LRRRR
LRRRL
LRRLR
LRRLL
LRLRR
LRLRL
LRLLR
LRLLL
LLRRR
LLRRL
LLRLR
LLRLL
LLLRR
LLLRL
LLLLR
LLLLL
(b)
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
🟦 Theory
🟥 Data
RR
RL
LR
LL
Prior 2 Outcomes

Figure 5


45 MIN
HIT
FALSE
ALARM
COUNTS/BIN
45 MIN