Tracing Eye Movement Protocols with Cognitive Process Models

Dario D. Salvucci (dario+@cs.cmu.edu)
Department of Computer Science; Carnegie Mellon University
Pittsburgh, PA 15213

John R. Anderson (ja+@cmu.edu)
Department of Psychology; Carnegie Mellon University
Pittsburgh, PA 15213

Abstract

In using eye movements to develop cognitive models, researchers typically analyze eye movement protocols with aggregate measures and test models with respect to these measures. Because aggregate analyses sometimes conceal informative low-level behavior, protocol analyses comparing model predictions to individual trial protocols are frequently desirable; however, protocol analysis for eye movement data is often tedious and time-consuming. We describe how to automate the protocol analysis of eye movements using hidden Markov models. Working with data from an equation-solving task, we demonstrate two methods of tracing eye movement data—that is, mapping eye movements to the sequential predictions of a cognitive process model. We evaluated these tracing methods in an experiment where participants were instructed to execute given equation-solving strategies. When coding the experimental protocols in terms of the given strategies, the automated tracing methods performed as well as human expert coders in a fraction of the time.

Introduction

In the cognitive sciences, the study of eye movements has become increasingly popular for the investigation of human problem-solving behavior. Eye movements provide numerous clues to underlying cognitive processes and their interactions with the outside world, helping to determine how and when people encode information, what information they use or ignore, and how they interleave encoding and computation. Many researchers have utilized eye movements to develop and test cognitive models in various domains. Typically, eye movements are analyzed in terms of aggregate measures—for instance, the number of fixations on an item or the total time spent fixating an item—and cognitive models are developed and tested with respect to these measures. This methodology has led to highly successful cognitive models in numerous domains, including reading (Just & Carpenter, 1980; Rayner, 1995) and arithmetic (Suppes, 1990).

While aggregate analyses help to understand aggregate behavior, they sometimes conceal additional informative aspects of behavior that appear in single trial protocols. Recognizing this problem, researchers often find it desirable to perform protocol analyses that compare model predictions directly to individual trial protocols, as is common with verbal protocols. Unfortunately, protocol analysis for eye movements is often extremely tedious. Several trials from even the simplest tasks can generate massive eye movement protocols which must be coded into a more convenient form for analysis. In addition, these protocols typically suffer from significant amounts of noise due to variability in both human scanning behavior and eye-tracking equipment. Because of the size and complexity of the data, it is often implausible for humans to analyze more than a few eye movement protocols in close detail without sacrificing consistency, accuracy, and large amounts of time.

Automated Eye Movement Protocol Analysis

This paper describes an automated approach to the analysis of eye movement protocols. Previous work on automated protocol analysis systems has concentrated primarily on verbal protocols (e.g., Waterman & Newell, 1971) and generic-action protocols (e.g., Ritter & Larkin, 1994). These systems use a process of *tracing* to map observed actions to the sequential predictions of a cognitive model (Ohlsson, 1990). Unfortunately, we have found that these systems do not generalize well to eye movement data, which can be collected at a very fine temporal grain size and often include significant individual and equipment variability. Our approach exploits the special characteristics of these data to allow for fast and accurate tracing of eye movement protocols. The proposed methods can analyze protocols in a fraction of the time needed by humans while achieving comparable, if not better, accuracy in their analyses.

The automated analysis of eye movement protocols has numerous significant applications in the real world. The methods can be used off-line to code larger, more complex eye movement data sets than human coders could manage. In addition, they can help evaluate fits of low-level cognitive models to large data sets at the level of trial protocols. The methods can also be used on-line to control eye-based input devices for user interfaces. They could also help intelligent tutoring systems disambiguate solution strategies which cannot be inferred solely from student responses.

Eye Movements and Hidden Markov Models

The proposed methods perform eye movement protocol tracing by means of hidden Markov models (HMMs). For years HMMs have been widely employed in implementing recognition systems for speech and handwriting. In many ways, the analysis of eye movements has much in common with speech and handwriting recognition. These recognition

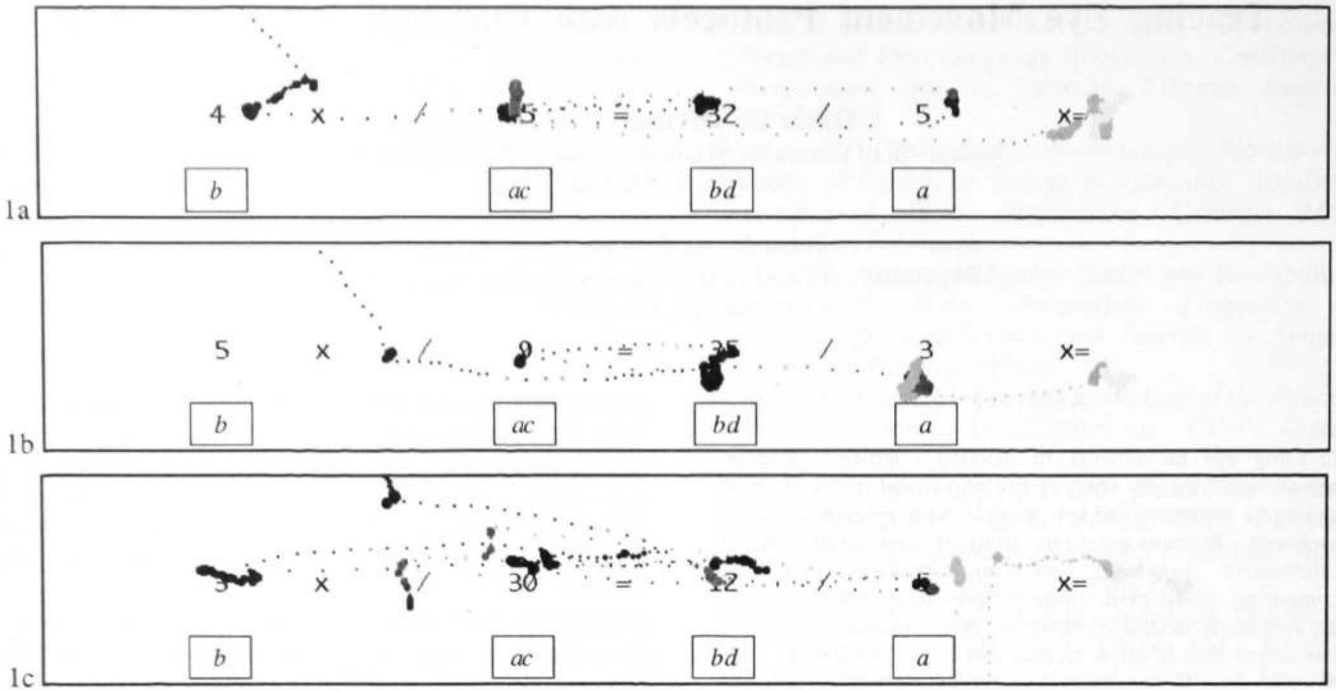

Figure 1: Sample equation-solving problem screens with eye movement protocols and target value labels.

systems take a person's speech or handwriting input and determine the most likely interpretation of this input given a model of the person's possible intentions. Our automated algorithms perform the analogous task for eye movements, taking a person's eye movements and determining the most likely sequence of intended fixations.

Several researchers have explored possibilities for applying HMMs, and more generally Markov models, to eye movement data. The most common use of Markov models has appeared in analyses of transition probabilities from one fixation target area to another (e.g., Stark & Ellis, 1981; Suppes, 1990). Although such analyses begin to shed light on the sequential nature of eye movement protocols, they ignore more global information by assuming that transitions do not depend on the prior sequence of fixations. Other researchers have used HMMs to represent velocity distributions in smooth eye movements (Kowler, Martins, & Pavel, 1984) and to model explicit foveal sequences (Rimey & Brown, 1991). We use ideas from such work as a basis for developing a more complete process model approach to tracing eye movements.

Tracing Eye Movements

Before discussing the details of the proposed methods for tracing eye movements with HMMs, we motivate these methods by illustrating the tracing process for a sample task and showing the difficulties that can arise in the process. Our sample task involves solving equations of the form:

$$b\ x\ /\ ac = bd\ /\ a$$

The terms a and b represent integers in the interval [2,9], and ac and bd represent the product of a and b with other integers c and d in [2,9], respectively. In the task, subjects must determine the value of the unknown quantity x, which in all cases equals the product cd. Thus, the process for solving the problems involves dividing ac by a to compute c, dividing bd by b to compute d, and multiplying c and d to compute the answer. For instance, for the sample problem

$$4\ x\ /\ 45 = 32\ /\ 5$$

the answer can be computed by finding $c = 45/5 = 9$, $d = 32/4 = 8$, and finally $cd = 9 \cdot 8 = 72$. For a single trial, eye movement data were collected while the subject solved the on-screen problem and typed her response into the answer box.

The tracing process requires a cognitive process model for the task which describes the steps taken in encoding items and computing results. The model may be implemented within a theory of cognition such as ACT-R (Anderson & Lebiere, 1998), but need only generate sequences of predicted actions for the task. Let us consider a sample model which can generate two distinct strategies for solving equations in the task: a left-to-right strategy (*b ac bd a*) and a paired-left-to-right strategy (*b bd ac a*). In the left-to-right strategy, items are encoded in order and intermediate results are computed at the earliest possible time—that is, d is computed after fixating bd and c is computed after fixating a. In the paired-left-to-right strategy, items are encoded in pairs as needed for intermediate results, decreasing working memory load. Both strategies assume that the structure of the equations has already been internalized, making fixations on the variable and operators unnecessary. Each strategy thus represents a cognitive process for solving the equations which manifests itself in a unique fixation ordering.

Using this sample model, we can trace a given protocol by mapping observed eye movements to predicted fixations. Figure 1a shows a sample problem screen for the above equation along with the subject's eye movement protocol, sampled every 8.3 ms; the protocol points are shaded and sized such that later samples appear lighter and lower-velocity samples appear larger. To trace the protocol with our sample model, we must map the observed data points to the best matching model strategy—in this case, the paired-left-to-right strategy (*b bd ac a*). In this mapping, the final two observed fixations on *ac* and *answer* cannot be mapped to the model since our simple model does not predict them. Note that the subject had already solved many similar problems and had learned this very efficient strategy for solving them.

Unfortunately, tracing eye movement protocols is rarely as simple as this example might imply. Figure 1b shows a similar protocol that exhibits the same (*b bd ac a*) strategy. However, the first fixation for this protocol lies ambiguously between several target areas; while humans may be able to resolve this ambiguity, a naive analysis algorithm that maps fixations to their closest targets might interpret the fixation as being on / or *ac* rather than its more likely interpretation, *b*. Figure 1c illustrates a protocol that presents serious difficulty for human and automated analysis alike; the protocol contains blinks and many ambiguous fixations that seriously degrade its readability. A successful automated analysis system must make intelligent global decisions to help resolve local ambiguities, making HMMs an excellent tool for tracing eye movements.

Tracing Eye Movements with HMMs

We now describe two automated methods for tracing eye movement protocols. The tracing algorithms take three inputs: *eye movement tuples*, *target areas*, and *model strategies*. The eye movement tuples comprise sampled points of the form $<x, y, v>$, where x and y indicate the location of the point and v indicates the velocity at that point; velocities can be calculated as adjacent point-to-point distances. The target areas include the name and location of possible fixation targets on the experiment screen; for the equation-solving task, the target areas would include the four values $\{b, ac, bd, c\}$, assuming we ignore possible fixations on the variable and operators. The model strategies comprise a set of possible fixation sequences predicted by some process model, such as the left-to-right and paired-left-to-right strategies discussed earlier. The tracing algorithms produce two outputs: a *model trace* and a *model evaluation*. The model trace represents a mapping from eye movement data points to the fixation sequence predicted by the best corresponding model strategy. The model evaluation represents the probability of the model trace, which can be used to evaluate the fit of the model to the data.

Both tracing methods trace eye movements by means of hidden Markov models (HMMs). HMMs are essentially probabilistic finite state machines: transition probabilities determine the likelihood of taking the transition from one state to the next, and observation probabilities determine the likelihood of seeing a particular observation in the state. More information on HMMs and their applications can be found in Rabiner (1989).

Saccade-Fixation Submodel (SFS) Tracing

The first tracing algorithm, saccade-fixation submodel (SFS) tracing, begins with the construction of a *saccade-fixation submodel* for each target area. Each submodel is itself an

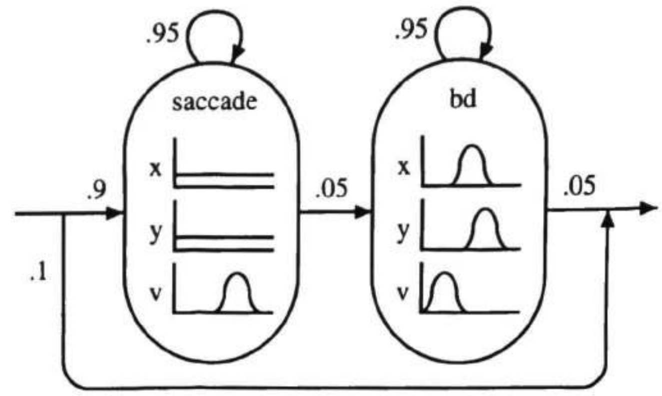

Figure 2: Saccade-fixation submodel for target *bd*.

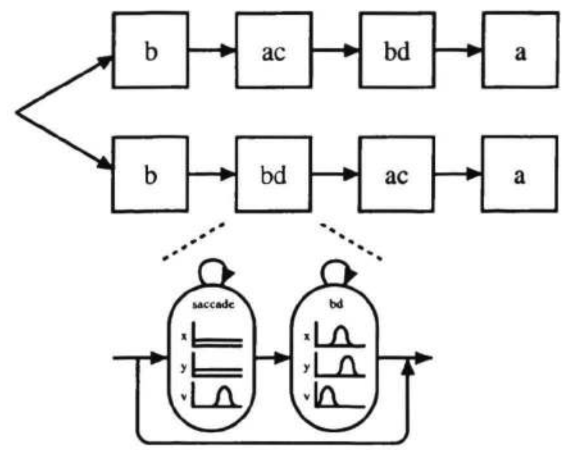

Figure 3: Sample SFS tracer model.

HMM which represents a model of the possible observations generated when a person intends to fixate that target. A sample saccade-fixation submodel for the *bd* target area is shown in Figure 2. The submodel has two states with three observation probability distributions: an x and y coordinate distribution and a v velocity distribution. The first state models points that represent saccades; its velocity distribution is weighted toward high velocities to model high-velocity saccadic movement, and its x and y distributions are uniform to show that saccades to this target can come from any location. The second state models points that represent fixations; its velocity distribution is weighted toward low velocities to model near-stationary fixations, and its x and y distributions have means over the center of the target area. Thus, as we attempt to find the most likely interpretation for a given protocol, high-velocity saccade points will likely match to the first state of the submodel, while low-velocity fixation points near the target area will likely match to the second state. The submodel's transition probabilities can be trained using the data set; however, we have simply chosen values that work well with our particular eye movement data.

Next, we use the submodels to construct a *tracer model* that incorporates the predicted fixation sequences of the model strategies. We use the model strategies to build the tracer model as follows. For each strategy, substitute each predicted fixation with its corresponding submodel and link the submodels serially. Then, link these strategy HMMs together in parallel to form the composite tracer model HMM. Figure 3 shows a sample tracer model for the left-to-right and paired-left-to-right strategies described earlier; the square boxes representing the predicted fixations are replaced by corresponding submodels, as shown.

Finally, given this tracer model, we can produce a model trace for a given protocol. We first determine the most likely state sequence for the protocol's eye movement tuples using the Viterbi HMM decoding algorithm (see Rabiner, 1989). Conceptually, this decoding process is analogous to laying out the tuple sequence onto the HMM such that the probability of the sequence (including observation and transition probabilities) is maximized. The decoded state sequence links the data points to predicted fixation targets, thus producing a model trace that maps the observed eye movements to the model's predictions. Note that the model trace describes both which model strategy was most likely executed and the most likely assignment of data points to their corresponding predicted fixations. The decoding process also provides the model evaluation value as the probability of the most likely sequence.

The primary cost of SFS tracing comes from decoding the eye movement protocol with the tracer model. The Viterbi decoding algorithm uses dynamic programming to decode sequences in $O(N^2T)$ time, where N is the number of HMM states and T is the length of the decoded sequence (see Rabiner, 1989). For SFS tracing, N depends crucially on the strategies predicted by the cognitive model—more strategies with longer sequences will increase the size of the tracer model and thus increase N. The length of the decoded sequence T corresponds to the length of the eye movement protocol being analyzed.

Centroid Submodel (CS) Tracing

SFS tracing, as we will soon see, works well in tracing various eye movement protocols. One problem with SFS tracing, however, is its speed: the number of possible states in the tracer model and the potential length of the point sequence can make SFS tracing somewhat slow. The second tracing algorithm, centroid submodel (CS) tracing, alleviates this problem by tracing eye movements in two stages: first, it finds the centroids of each fixation in the protocol; and second, it generates a model trace by mapping the fixation centroids onto the model's predicted fixations. Though this two-stage process is faster, it has the disadvantage of incurring a loss of information between the two stages: while the SFS tracer model can influence where fixations are identified and to which targets they correspond, the CS tracer model can only influence to which targets the given centroids correspond. We will discuss this tradeoff further in the next section.

The first step of CS tracing involves producing a sequence of fixation centroids for the given protocol. We perform this task using an HMM similar to the SFS-tracing saccade-fixation submodels, except without positional x and y distributions. The HMM, shown in Figure 4, has a saccade state and a fixation state as before, but the two states are linked together to form a circular path between them. Thus, this HMM models repeated saccades and fixations around the

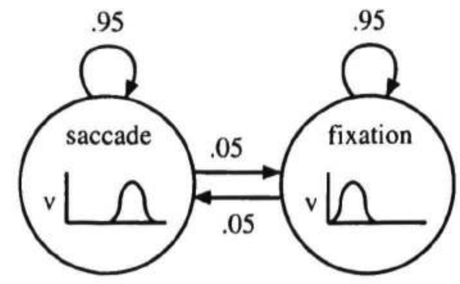

Figure 4: Centroid decoding HMM.

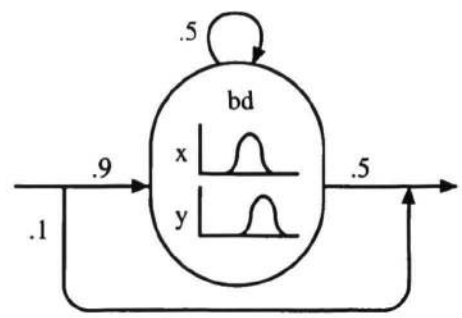

Figure 5: Centroid submodel for target *bd*.

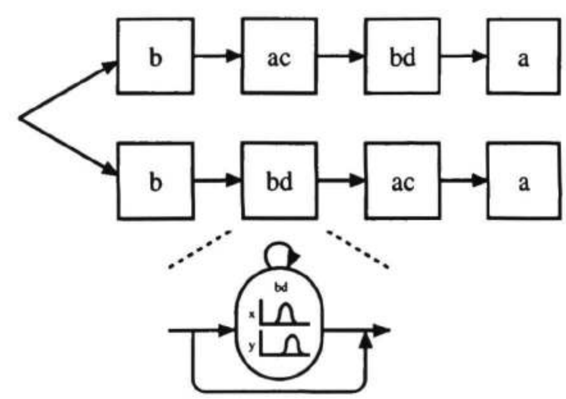

Figure 6: Sample CS tracer model.

screen without making predictions about where these actions occur. The parameters shown in Figure 4 work well for our particular eye movement data; in general, however, both observation and transition probabilities can be learned using HMM parameter reestimation over a set of training protocols (see Rabiner, 1989).

Given this HMM, we produce the desired centroid sequence as follows. First, we create a sequence of velocity-only point data by removing location information from the eye movement tuple data. Next, we decode the most likely state sequence through the HMM. Finally, we take the resulting state sequence, remove saccade points corresponding to the first state, and collapse fixation points corresponding to the second state. As we collapse the fixation points, we calculate the centroid of each fixation (using location information from the original point data), yielding the desired sequence of fixation centroids.[1]

[1] There are various other methods of separating fixations and saccades—for instance, using a strict velocity cutoff or moving

After computing fixation centroids, we now generate a model trace by mapping the centroid sequence onto the predictions of the model. The process of finding a model trace in CS tracing is identical to that in SFS tracing except for the submodels used. Instead of saccade-fixation submodels, CS tracing uses *centroid submodels* that model the distribution of where centroids can occur for each target area. Figure 5 illustrates one possible submodel for the *bd* target, with the means of the x and y distributions over the center of the *bd* target area. To account for skipped or extra fixations, the submodel allows the state to be bypassed or repeated with the given probabilities. Once the centroid submodels are constructed, we proceed to produce a tracer model and decode the most likely model trace as before. A sample tracer model with simple centroid submodels is illustrated in Figure 6.

Because CS tracing uses the Viterbi algorithm in both stages of analysis, the complexity of CS tracing is the same as that of SFS tracing, namely $O(N^2T)$. In practical application, however, the two stages of CS tracing can run much faster than SFS tracing. In the first stage, CS tracing uses a two-state HMM to decode fixations and find fixation centroids. Although T, the length of the decode sequence, may be large, N remains small, namely $N = 2$. In the second stage, CS tracing decodes centroids using its tracer model. Here, N may be large (depending on the model strategies), but T is much smaller than in SFS tracing, since T represents the number of fixation centroids rather than the number of data points. Thus, both stages of CS tracing can operate more efficiently than the one stage in SFS tracing.

Testing the Tracing Methods

The SFS and CS tracing methods provide an automated mechanism for mapping eye movements to their underlying thought processes. Because we can never be sure what a person was thinking while executing a particular eye movement pattern, we cannot possibly describe the "accuracy" of these methods with absolute certainty. We can, however, provide strong support for the methods in two ways: evaluating the methods on protocols collected in what we call an "instructed-strategy" paradigm, and comparing the methods' interpretations of protocols to that of human expert coders.

The instructed-strategy paradigm has subjects executing a specific strategy when solving a task. Before starting the task, the subject is instructed to solve the task using a specific strategy that describes a strict order of accessing information and computing intermediate results. Of course, we cannot guarantee that the subject perfectly executes the given strategy, but we make it clear to subjects that they are to use the strategy as diligently as possible. Using this paradigm, we can show support for the tracing methods by comparing their interpretations to the subjects' given strategies; the paradigm gives us a "correct" interpretation with which we can test the tracing algorithms.

Because subjects may not execute the given strategies as instructed, it is reasonable to expect that the tracing methods will not interpret some number of the protocols "correctly." However, since the tracing methods automate an analysis task that is typically performed by humans, our primary interest is to know how well the tracing methods perform in comparison with human coders. Our goal is to have the tracing methods interpret protocols as well as or better than human coders.

Data Collection and Human Coding

Using the instructed-strategy paradigm, we collected protocols from five Carnegie Mellon students performing the equation-solving task. Each subject solved 24 problems per session for five sessions. On the first session, subjects were given the equations with no instruction and asked to solve for the unknown quantity. On each of the four following sessions, subjects were given a single strategy and asked to execute this strategy as faithfully as possible in solving the problems. The four strategies given to subjects included the order with which to fixate equation elements and the time at which to compute intermediate results. The fixation orderings, representative of the strategies entailed by these orderings, were: left-to-right, (*b ac bd a*); paired-left-to-right, (*b bd ac a*); right-to-left, (*a bd ac b*); and paired-right-to-left, (*a ac bd b*). Due to extreme noise in the protocols, data from one subject were omitted from analysis.

We gave a subset of the collected protocols to human expert coders to compare their performance to that of the tracing methods. The test protocols comprised the last two protocols from each session for each of the four subjects (32 protocols in all) and were given to the coders in displays similar to those in Figure 1, with additional information describing the sequence of fixations. The human coders then classified each protocol as one of the four given strategies. Both human coders (a professor and graduate student in the cognitive sciences) were highly experienced in examining such displays and in working with cognitive process models.

Results

We ran both tracing methods on all the protocols and had them determine which of the four given strategies best fit each protocol. We first compare the tracing methods' performance to that of human coders on the 40 test protocols. We then evaluate the tracing methods over the entire data set with respect to the given instructed strategies.

Table 1 shows the percent agreement between the classifications of the two tracing methods (SFS and CS), the classifications of the two human coders (HC1 and HC2), and the given strategies for all subjects. The tracing algorithms show the highest agreement with the given strategies (90.6-93.7%), while the human coders exhibit somewhat lesser agreement (78.1-90.6%). Both human coders show predominantly equal or higher agreement with the tracing methods (HC1-CS, 81.2%; HC2-CS, 87.5%; HC2-SFS, 90.6%) than with each other (HC1-HC2, 81.2%). Also, while the first human coder agrees more with CS tracing than SFS tracing, the second agrees more with SFS tracing. In summary, performance for the tracing methods is almost indistinguishable from human coder performance.

windows with a given dispersion. We use the two-state HMM method to illustrate similarities with SFS submodels.

Table 1: Percent agreement between tracing methods, human coders, and given strategies.

	CS	SFS	HC1	HC2	Given
CS	100.0	96.9	81.2	87.5	90.6
SFS		100.0	78.1	90.6	93.7
HC1			100.0	81.2	78.1
HC2				100.0	90.6

Table 2: Average time to code a protocol, in seconds.

CS	SFS	HC1	HC2
0.2	5.8	67.5	60.0

The above results show the similarity between tracing method and human coder performance but, because of the small size of the test set, they say little about how the tracing methods compare to each other. To answer this question, we evaluated each method over the entire data set for the second through fifth sessions. Both tracing methods performed extremely well, with SFS tracing performing only slightly better: SFS tracing agreed with 93.8% of the given strategies, while CS tracing agreed with 91.3%. The loss of information in CS tracing between its two stages—namely, the strict decision in the first stage as to what is a fixation—does not seem to sacrifice performance significantly; that is, deciding to which targets fixations correspond is more important than deciding where fixations occur.

While the tracing methods and human coders performed similarly in tracing the test protocols, the tracing methods completed the task significantly faster. Table 2 shows the average time in seconds needed to code one protocol for each tracing method and human coder, where tracing method times were collected on a 200 MHz Power Macintosh. While the human coders required approximately one minute per protocol, the tracing methods performed at least an order of magnitude faster: SFS tracing coded the protocols approximately 10 times faster and CS tracing coded them approximately 300 times faster.

Conclusions and Future Work

Overall, the results for SFS and CS tracing on the equation-solving protocols are very encouraging. Both tracing methods can successfully analyze protocols in a fraction of the time needed by human expert coders. The resulting traces and evaluations can be used by researchers to explore many aspects of task behavior, including frequencies of strategy use, meta-strategies over time, and process model fits to data. The tracing methods' speed also makes them amenable to real-time applications such as intelligent tutoring systems and eye-based input devices.

This study only addresses a few of the vast possibilities for analyzing eye movements with HMMs. We are now investigating some of these other possibilities, including:

- creating more efficient HMMs using model strategies with a hierarchical subgoal structure
- incorporating time information such that submodels predict both fixation duration and location
- optimizing submodel probabilities by means of HMM parameter reestimation

We also plan to apply these tracing techniques to different task domains in an effort to evaluate their usability as general-purpose sequential data analysis tools.

Acknowledgments

This work was supported in part by a National Science Foundation Graduate Fellowship awarded to Dario Salvucci and Office of Naval Research grant N00014-96-1-0491 awarded to John Anderson.

References

Anderson, J. R., and Lebiere, C. (1998). *The Atomic Components of Thought*. Hillsdale, NJ: Erlbaum.

Just, M. A., & Carpenter, P. A. (1980). A theory of reading: From eye fixations to comprehension. *Psychological Review, 87*, 329-354.

Kowler, E., Martins, A. J., & Pavel, M. (1984). The effect of expectations on slow oculomotor control IV: Anticipatory smooth eye movements depend on prior target motions. *Vision Research, 24*, 197-210.

Ohlsson, S. (1990). Trace analysis and spatial reasoning: An example of intensive cognitive diagnosis and its implications for testing. In N. Frederiksen, R. Glaser, A. Lesgold, & M. G. Shafto (Eds.), *Diagnostic Monitoring of Skill and Knowledge Acquisition*. Hillsdale, NJ: Erlbaum.

Rabiner, L. R. (1989). A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE, 77*, 257-286.

Rayner, K. (1995). Eye movements and cognitive processes in reading, visual search, and scene perception. In J. M. Findlay, R. Walker, & R. W. Kentridge (Eds.), *Eye Movement Research: Mechanisms, Processes, and Applications*. New York: Elsevier Science Publishing.

Rimey, R. S., & Brown, C. M. (1991). Controlling eye movements with hidden Markov models. *International Journal of Computer Vision, 7*, 47-65.

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

Stark, L., & Ellis, S. R. (1981). Scanpath revisited: Cognitive models of direct active looking. In D. F. Fisher, R. A. Monty, & J. W. Senders (Eds.), *Eye Movements: Cognition and Visual Perception*. Hillsdale, NJ: Erlbaum.

Suppes, P. (1990). Eye-movement models for arithmetic and reading performance. In E. Kowler (Ed.), *Eye Movements and their Role in Visual and Cognitive Processes*. New York: Elsevier Science Publishing.

Waterman, D. A., & Newell, A. (1971). Protocol analysis as a task for artificial intelligence. *Artificial Intelligence, 2*, 285-318.