Mapping Eye Movements to Cognitive Processes

Dario D. Salvucci
May 10, 1999
CMU-CS-99-131

Department of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy

Thesis Committee:
John R. Anderson, Chair
Herbert A. Simon
Kenneth R. Koedinger
Keith Rayner, UMass Amherst

This research was sponsored by the National Science Foundation (NSF) through a generous grant and subgrant to Carnegie Mellon University. The views and conclusions contained in this document are those of the author and should not be interpreted as representing the official policies, either expressed or implied, of the NSF or the U.S. government.

Keywords: tracing, cognitive models, eye movements, hidden Markov models, user models, protocol analysis, sequential data analysis.

Abstract

Eye movements provide a rich and informative window into a person's thoughts and intentions. In recent years researchers have increasingly employed eye movements to study cognition in psychological experiments, to understand behavior in user interfaces, and even to control computers through eye-based input devices. Unfortunately, like speech and handwriting, eye movements generate vast amounts of data with significant individual variability and equipment noise. Thus, the analysis of eye-movement data—that is, determining what people are thinking based on where they are looking—can be extremely tedious and time-consuming. Typical eye-movement data sets are simply too large and complex to be analyzed by hand or by naive automated methods.

This thesis formalizes a new class of algorithms that provide fast and robust analysis of eye-movement data. Specifically, the thesis describes three novel algorithms for tracing eye movements—mapping eye-movement protocols to the sequential predictions of a cognitive process model. Two algorithms, fixation tracing and point tracing, employ hidden Markov models to determine the best probabilistic interpretation of the data given the model. The third algorithm, target tracing, extends an existing tracing algorithm based on sequence matching to eye movements. The thesis also formalizes several algorithms for identifying fixations in raw eye-movement protocols and provides a working system, EyeTracer, that embodies the proposed tracing and fixation-identification algorithms.

To demonstrate the power of the proposed algorithms, the thesis applies them in three real-world domains: equation solving, reading, and eye typing. The equation-solving studies show how the algorithms can code, or interpret, eye-movement protocols as accurately as expert human coders in significantly less time. The studies also illustrate how the algorithms facilitate the prototyping and refinement of cognitive models. The reading study demonstrates how the algorithms help to evaluate and compare two existing computational models of reading and clear up temporal aspects of reading data using sequential aspects of the data. The eye-typing study shows how the algorithms can interpret eye movements in real time and help eliminate usability restrictions imposed by existing eye-based interfaces.

Acknowledgments

I am deeply indebted to so many family and friends for all their love and support throughout these graduate years. This work would not be possible without their constant guidance and inspiration and I offer them my dearest thanks.

My thesis advisor and committee provided continuous support during every stage of this endeavor. John Anderson inspired countless new ideas and challenged me to strive for my best possible work. His tireless help was invaluable and I am fortunate to have had him as a mentor and friend. Ken Koedinger sparked numerous stimulating discussions about both the thesis work and a vast array of other interesting topics. Herb Simon helped greatly in shedding new light on the work and understanding its relation to larger issues. Keith Rayner offered many insights into the work and provided a great deal of guidance and encouragement.

A number of people at Carnegie Mellon were influential in my graduate life. The ACT-R research group, especially Kevin, Frank, Scott, Lisa, Mike, Alex, Christian, Marsha, Dan, Mike, and Albert, helped in the development of this work and made Carnegie Mellon an enjoyable environment. Adisack, Chris, Lynne, and other Cog Squad members joined in on tennis matches on many summer afternoons. Helen and Janet were always willing to lend a helping hand with even the most trivial problems. Bonnie, Thomas, Ken, and John helped immensely in the job application process.

My friends both in and out of the Pittsburgh area have supported me throughout and helped me to maintain a healthy perspective on life. Perry, Doug, Lars, Merete, Carsten, and Molly filled my graduate years with fun and happy times, including tennis matches, golf outings, and Super Bowl parties. Alex, Michael, Anthony, Maria, and Myriam provided lighthearted support from afar and many great excuses to take time off and travel.

I owe two very special debts of gratitude. Ayla has stood by me through all five years of ups and downs, always willing to help, always encouraging me, always believing in me. Her steadfast love, support, and dedication have been a tremendous inspiration, and for that and all that she has done for me, I am eternally grateful.

I am also deeply indebted to my immediate and extended family for all their love and guidance during my graduate years. Mom and Dad have sacrificed so much to give me an opportunity to fulfill my dreams. Silvia has been inspirational in appreciating and enjoying the little things in life. Claudio has been a constant reminder to do what you love and to love what you do. None of this would have been possible without their tireless love and support.

Table of Contents

Chapter 1

Introduction

Windows to the Mind

Our eyes reveal a great deal about us; whether or not our eyes are "windows to the soul," as the common saying goes, they are certainly windows to the mind. A teacher watches her students' eyes for signs of attention and understanding; a linebacker watches a quarterback's eyes for clues to his intended receiver; a poker player watches his opponent's eyes for any indication of bluffing. Whenever we interact with others—in the classroom, on the playfield, wherever—we examine their eye movements to help discern what they are thinking. Eye movements make up an extremely informative component of human observable behavior that, along with other components like speech and gesture, allow us to infer what people are thinking based on their actions.

The connection between eye movements and thoughts has enjoyed burgeoning attention in recent years in a number of major research areas. In psychology and the cognitive sciences, researchers have studied eye movements to help elicit and analyze the cognitive processes in a variety of task domains. This research has provided notable and significant benefits in domains such as reading (e.g., Just & Carpenter, 1980, 1984; Schilling, Rayner, & Chumbley, 1998), arithmetic (Suppes, 1990), and word problems (e.g., Hegarty, Mayer, & Green, 1992), to name a few. In neurobiology and vision research, researchers have focused on the underlying mechanisms of the ocular system and low-level aspects of eye movements (e.g., Erkelens & Vogels,1995; Fuchs, 1971; Vaughan, 1982; Zingale & Kowler, 1987). This work has laid a solid foundation for understanding fundamental characteristics of human visual processing. In human-computer interaction, researchers have studied eye movements to improve the speed and usability of user interfaces. Such work has resulted in better understanding of interface use (e.g., Aaltonen, Hyrskykari, & Räihä, 1998) and successful eye-driven user interfaces (e.g., Hutchinson et al.,1989; Stampe & Reingold, 1995).

There are a number of important reasons why eye movements have become so popular in so many research fields. Eye-movement protocols—sequences of recorded eye-gaze locations—represent actions at a fine temporal grain size, typically on the order of 10 milliseconds. At this grain size, eye movements yield important clues to human behavior, including what information people use in problem solving and when they use it; how much time people need to process various pieces of information; and when people forget and review previously encoded information. Also, humans need little if any instruction or training to produce informative data; in most applications, eye-movement data is collected non-intrusively, such that data collection in no way affects task performance. In addition, eye movements can serve as the sole source of data or as a supplement to other sources like verbal protocols. Thus, while eye movements certainly do not entirely reveal a person's thoughts, their flexibility and informativeness make them an excellent data source for many studies and applications.

Let us briefly illustrate the informativeness of eye movements with two sample protocols. Figure 1.1 shows a sample protocol for an equation-solving task where subjects solved equations of the form $b\,\mathbf{x}\ /\ ac = bd\ /\ a$. For each trial, subjects mentally computed the solution value of $\mathbf{x} = cd = (ac/a)(bd/b)$ and typed the value into a response field. The eye-movement protocol appears as a sequence of sampled point-of-regard data points, where later samples appear lighter in the protocol. The protocol shows the alternating pattern of fixations (pauses while the eye encodes some information, shown as large points) and saccades (rapid movements between fixations, shown as small points) present in so-called saccadic eye movements. This protocol reveals a great deal about this subject's thought processes during the trial. From the start point (fixation 1, not shown), the subject encodes $ac = 40$ (fixation 2) and $a = 5$ (fixation 3) and presumably computes the intermediate value $c = 8$. Next, the subject encodes $b = 3$ (fixation 4) and $bd = 27$ (fixation 5) and again presumably computes the intermediate value $d = 9$. Finally, the subject computes the solution $\mathbf{x} = 72$ and types this response to end the trial. The subject, who has completed a number of trials prior to this one, has clearly learned a very efficient strategy for solving these problems. While coarser data such as reaction time or correctness may help reveal such strategies, eye movements elicit the individual steps of the problem-solving process at a much finer-grain level of detail. They also allow for non-intrusive data collection, unlike verbal protocols, that is essential in tasks with short response times.

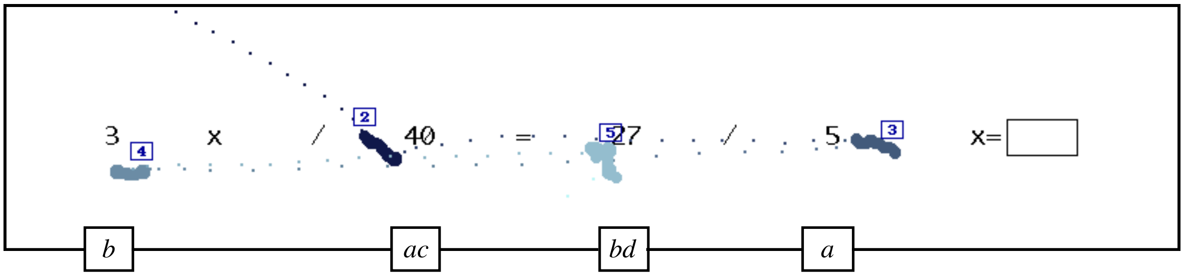


Figure 1.1: Sample equation-solving protocol.

While the above protocol provides a relatively clear window into the subject's thought processes, protocols for many tasks reveal aspects of cognition in much subtler ways. Figure 1.2 shows a sample eye-movement protocol for a task in which subjects simply read a given sentence. While the protocol plainly reveals a straightforward left-to-right reading scan, deeper understanding of what the subject is thinking at each fixation is far from obvious: the fixations do not occur at even intervals, nor do they occur over every word, nor do their durations reveal any clear pattern. Nevertheless, eye-movement research in the reading domain, by focusing primarily on the frequency and duration of fixations on words, has resulted in significant advancements in our knowledge of reading; for instance, this research has demonstrated effects of word frequency on gaze durations and preview effects of viewing words in the periphery (see Rayner, 1995). Thus, while the clues provided may not always be obvious at first glance, detailed and rigorous study of eye movements in many tasks can reveal a variety of subtle yet interesting patterns in human behavior.

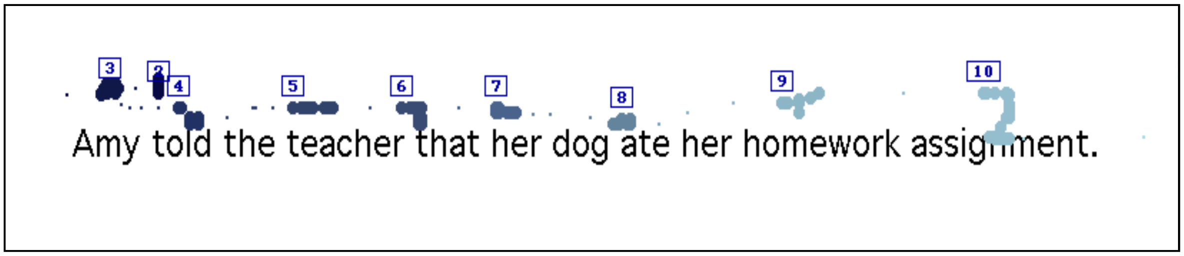


Figure 1.2: Sample reading protocol.

Eye-Movement Data Analysis

All research using eye movements, whatever the context, shares the need to analyze—or interpret—eye movements. As with other types of data, such as mouse clicks or verbal protocols, eye-movement data represent only clues to a person's thoughts; some analysis is always required to infer these thoughts from observed data. The interpretation derived by data analysis thus maps the observed eye-movement data to some predicted sequence of thoughts—that is, maps where the person was looking to what they were thinking. Of course, there are never guarantees as to whether some interpretation is correct, but we can often estimate the likelihood of different interpretations based on particular properties of the task environment combined with our understanding of the human visual and cognitive systems.

Difficulties in Data Analysis

Eye-movement data analysis reveals substantial information about human thoughts. Unfortunately, this analysis can often be difficult, tedious, and time-consuming. There are at least two major reasons for this difficulty. One reason is the sheer amount of eye-movement data in typical applications. Because they are collected at a fine temporal grain size, eye-movement data sets can become prohibitively large for even simple tasks. These massive data sets are typically translated into a more compact form for analysis, sometimes with significant loss of information. Researchers must often trade off the amount of informativeness extracted from eye-movement data with the amount of time needed to analyze all the information.

A second major reason for the difficulty of analyzing eye movements arises from noise due to individual and equipment variability. In many tasks individuals show great variability in their eye movements even when they are performing approximately similar strategies. Part of this variability arises from reviewing previously encoded information: Because reviewing often has little cost besides one or two extra fixations, individuals frequently review information they are unsure of or may have forgotten. Another contributor to individual variability involves errors in actual eye movements, such as the undershoot or overshoot of an intended fixation target. Finally, individuals can vary significantly in how and when they utilize peripheral (or parafoveal) encoding of information. Equipment variability also contributes to the (sometimes significant) noise in eye-movement data. Current eye trackers have a typical resolution ranging from 1/4° to 1° of visual angle depending on the particular equipment used to estimate point-of-regard location. While this resolution serves well for many applications and eye-tracker technology continues to advance, interpretation of eye-movement protocols must always take into account possible variability in the recorded data. Calibration problems, where entire trial protocols are shifted by some constant bias, frequently cause misinterpretations and require special techniques to avoid or correct.

We can illustrate a number of these issues with further sample eye-movement protocols. Figure 1.3 shows another equation-solving protocol in which the subject encodes the four numbers in the equation. In this protocol, fixation 2 appears midway between the two leftmost numbers 5 and 9. Interpreting this off-target fixation can be problematic: A naive interpretation that simply maps fixations to their nearest target might interpret fixation 2 as the encoding of 5 or 9, or perhaps even **x** or /. However, if we take the entire protocol into account along with other protocols we have seen, we note that each of the three numbers besides 5 has a fixation directly over it, and that after many trials subjects do not encode the operators or the equal sign. Using these observations, we conclude that the most likely interpretation of the protocol maps fixation 2 to an encoding of the number 5. In this case, the off-target fixation can primarily be attributed to individual variability as either an overshoot of the target or possibly an intentional parafoveal encoding; since the other fixations appear directly over their targets, equipment noise for this protocol seems less a factor. This protocol illustrates that naive analyses can sometimes be inadequate and that we may need to make use of various types of information to form sensible interpretations.

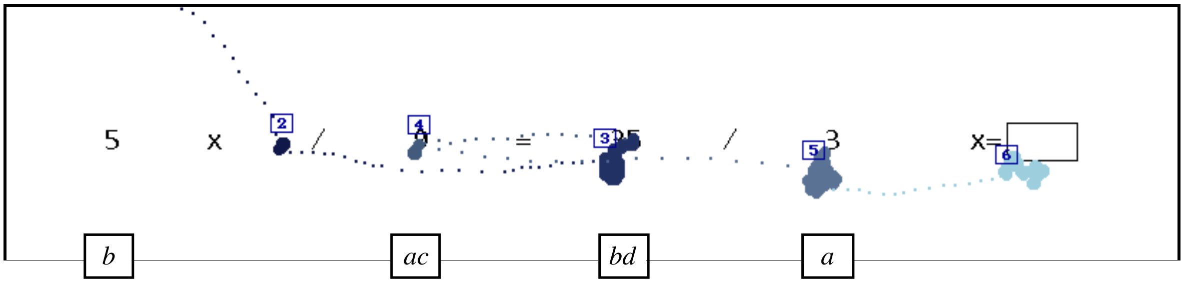


Figure 1.3: Sample equation-solving protocol.

Figure 1.4 shows another sample protocol taken from a task in which subjects type words by looking at letters on an on-screen keypad. The Figure 1.includes the keypad portion of the screen along with a user's eye movements when typing the word *SQUARE*. The user first fixated the word to type (fixation 1, not shown) and then fixated the letters *S* (fixation 2) and *Q* (fixation 3). After undershooting the next target with a fixation near *Y* (fixation 4), the user fixated the letter *U* (fixation 5). The user then fixated the letter *A* (fixation 7) with an intermediate fixation near *E* along the way (fixation 6). Finally, the user fixated the letters *R* (fixation 8) and *E* (fixation 9). This protocol nicely illustrates how variability can cause confusion in interpretation. The two incidental fixations (fixations 4 and 6) would be included in a naive interpretation as the letters *Y* and *E*. Also, off-center fixations that occur between targets could be interpreted as either target; for instance, the fixation intended for *Q* (fixation 3) would be mapped to its actual closest target *A*. Again we see that simple approaches to data analysis cannot adequately handle the complexity of typical eye-movement protocols.

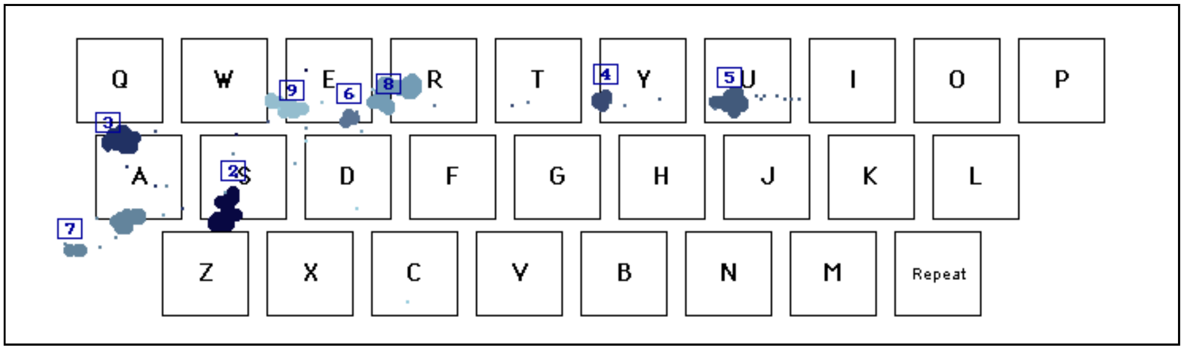


Figure 1.4: Sample eye-typing protocol for the word *SQUARE*.

Automated Data Analysis

The difficulty and tedium of analyzing eye movements often make it implausible for humans to analyze protocols fully by hand; it is simply too difficult to analyze the data consistently, accurately, and in a reasonable time frame. Thus, a number of researchers have aimed to develop automated tools to assist in eye-movement data analysis (e.g., Cabiati et al., 1983; Karsh & Breitenbach, 1983), an approach that has achieved some success for verbal protocols (e.g., Waterman & Newell, 1971, 1973). These automated tools have served many uses, primarily identifying fixations and saccades (e.g., Cabiati et al., 1983; Widdel, 1984) and assigning fixations to intended targets (e.g., Salvucci & Anderson, 1998). Automated tools allow investigators to analyze larger, more complex data sets in a consistent, detailed manner that would otherwise be impossible. Of course, "automated" data analysis is never fully automated: Humans always participate in some ways, such as defining relevant regions of interest, deciding what aspects of the data are important, or building predictive process models of the task. Thus, automated data analysis can be more accurately characterized as a semi-automated, interactive analysis process in which automated tools assist human investigators in performing faster, more accurate, and more consistent analyses.

Automated data analysis tools, while convenient, can be complex to design and implement: The same difficulties that plague human analysis of eye-movement protocols affect automated methods as well. The variability in these protocols requires that such tools form flexible interpretations that handle off-target, missing, and incidental fixations. In addition, while computers can usually analyze data significantly faster than humans, there remain serious speed-accuracy tradeoffs that tool developers must address. Automated analysis tools must be designed with care to provide sensible and accurate interpretations in the face of these difficulties.

Tracing Eye Movements with Cognitive Process Models

This thesis explores a new class of algorithms that analyze eye movements using a rigorous form of protocol analysis called *tracing*. Tracing is the process of mapping observed action protocols to the sequential predictions of a cognitive process model. Tracing has become an extremely useful tool for analyzing data in many contexts. Cognitive scientists have employed tracing to analyze verbal protocols in various problem-solving domains (e.g., Newell & Simon, 1972). Builders of intelligent tutoring systems have used tracing to determine a student's solution path through a student model of the learning domain (e.g., Anderson et al., 1990). Researchers in human-computer interaction have employed tracing to study the fits of user models (e.g., Card, Moran, & Newell, 1983) and to develop intelligent interfaces that analyze and react to user actions.

Tracing eye movements requires a cognitive process model capable of predicting sequences of eye movements performed during a task. This process model may be implemented in a number of ways, such as a production system (e.g., Anderson & Lebiere, 1998; Just & Carpenter, 1992; Meyer & Kieras, 1997; Newell, 1990) or a cognitive grammar (e.g., Smith, Smith, & Kuptsas, 1993). Of course, because this thesis specifically addresses eye movements, the model should predict visual processing at approximately the level of individual fixations or gazes. For example, consider the following process model for the equation-solving task (as shown in Figures 1 and 3), where subjects solve equations of the form $b\,\mathbf{x} \,/\, ac = bd \,/\, a$:

- encode *b*, encode *ac*, encode *bd*, encode *a*, compute *c*, compute *d*, compute **x**
- encode *b*, encode *bd*, compute *d*, encode *ac*, encode *a*, compute *c*, compute **x**

This process model allows two possible strategies for solving the problem. The first strategy encodes the four problem values from left to right, performing the three necessary computations after encoding all values. The second strategy encodes the values in pairs and computes intermediate values as soon as possible.

Given such a process model, tracing maps an eye-movement protocol to the best-fitting process model strategy. Figure 1.5 shows the mapping of a sample protocol to the process model above. The arrows in Figure 1.5 represent the trace, or interpretation, of the protocol—that is, the mapping between predicted fixations and their corresponding observed fixations in the protocol. In this case, the protocol best matches the second strategy that encodes values in pairs. From the start position (fixation 1, not shown), the subject jumps to the equation but misses the first value (fixation 2). The subject then encodes *b* (fixation 3), *bd* (fixation 4), *ac* (fixation 5), and *a* (fixation 6). Finally, the subject reviews *ac* (fixation 7) and fixates the output box (fixation 8). Note that the mapping includes only predicted fixations (fixations 3-6); unpredicted fixations (fixations 2, 7-8) and actions that do not correspond to observed fixations (the *compute* actions) are omitted. As the example illustrates, tracing takes advantage of both global and local information in forming interpretations of protocols. Local information, namely the location and velocity of each data point, helps separate fixations and saccades and pinpoint the locations of each fixation. Global information, namely the sequence of fixations, helps sort out the assignment of fixations to their intended targets.

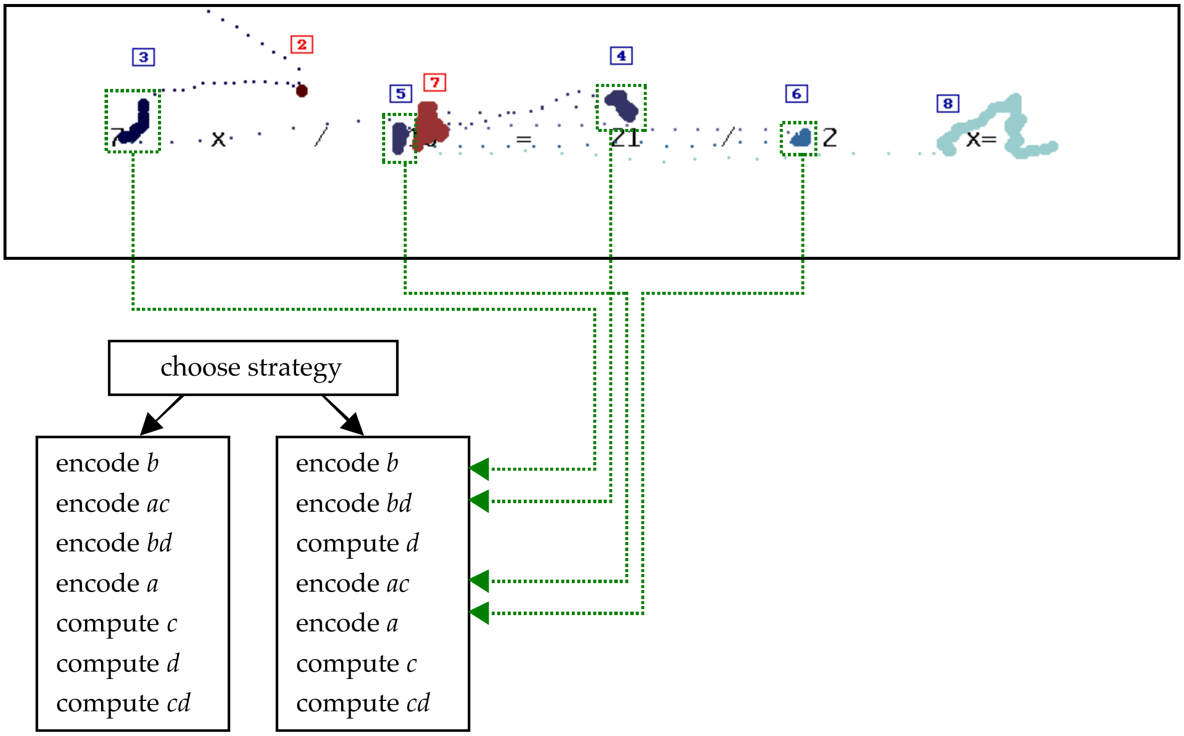


Figure 1.5: Sample equation-solving protocol, model, and trace.

Tracing eye movements has a number of applications in cognitive and computer science research. One important application for eye-movement tracing is in prototyping, evaluating, and refining cognitive models. For instance, Ritter and Larkin (1994) developed a methodology for developing cognitive models called trace-based protocol analysis, which prescribes an iterative process of tracing, trace analysis, and model refinement. This methodology, in conjunction with eye-movement tracing, allows for both exploratory and confirmatory analysis of eye-movement protocols. For exploratory analysis, a researcher can reduce a raw eye-movement protocol to fixation sequences without employing any model bias. The fixation sequences can then be examined for common patterns which form the basis of a prototype model. For confirmatory analysis, the researcher can trace the protocols with the prototype (or later) model to find mismatches between observed data and model predictions. The mismatches provide useful clues in pinpointing weaknesses of the model and eliminating them in future model refinements. In addition, tracing can provide a goodness-of-fit measure that helps compare the accuracy of different models in predicting a given protocol data set.

A related application of eye-movement tracing is the automated coding of eye-movement protocols from psychological experiments or user interactions. Protocol coding, especially common for verbal protocols, involves interpreting a protocol and classifying it as one of a fixed number of strategies or behaviors. Tracing classifies protocols by identifying the best-fitting model sequence for each protocol, as illustrated in the Figure 1.5 example. The resulting classifications can be used to examine frequencies of various strategies, to study within-subject strategy variations, and to identify meta-strategies. Automated coding is especially important for eye movements, as for verbal protocols, because of the size, variability, and complexity of typical data sets.

Another important application of eye-movement tracing arises in on-line systems that interpret eye movements as they are being generated. As opposed to the off-line nature of trace-based protocol analysis and automated coding, on-line tracing is useful for user interfaces that react to user actions as they occur. For instance, intelligent tutoring systems typically trace student actions to infer their current state of knowledge. In such systems, eye movements help disambiguate problem-solving strategies which cannot be inferred solely from more standard data (i.e., typing and mouse actions). As another example, user interfaces with eye-based input devices allow users to execute commands with explicit or implicit eye movements. Better eye-movement tracing algorithms allow such interfaces to better interpret user intentions and thus provide more efficient, usable interaction.

Thesis Overview

This thesis explores a new class of algorithms for tracing eye-movement protocols. Tracing eye movements encompasses a wide range of potential methods and issues. To restrict the scope of this work to a manageable subset, the thesis makes the following assumptions:

- The eye-movement protocol consists of a sequence of point-of-regard data, each of which gives the planar coordinates of the estimated look point on a computer screen. We assume that the point data are sampled at a constant rate, which depends on the specific eye-tracking equipment used.

- The cognitive process model using in tracing is a model capable of generating one or more predicted action sequences. We assume that the model predicts visual processing at the level of fixations or gazes. In addition, we allow the model to be non-deterministic, such that each possible predicted sequence occurs with some probability.

- The task environment in which eye-movement data are collected is (at least for the most part) static, with few if any changes occurring during any individual trial. The display should have easily-defined visual targets, such as numbers or words, that roughly translate to rectangular regions of interest. (The issues of tracing in dynamic environments and inferring regions of interest are addressed in the final chapter.)

The thesis comprises three main components. First, the thesis formalizes a methodology for tracing eye movements that includes both completely novel algorithms and extensions of previous ones. Second, it demonstrates the usefulness of these methods by applying them in three representative domains. Third, it provides a set of recommendations for their use and discusses a number of possible future extensions. The following sections outline these components.

Methodology and Implementation

The first component of the thesis formalizes and describes a set of methods that automate the tracing of eye-movement protocols. Chapter 3 details a number of existing and novel algorithms for fixation identification. Chapter 4 describes three novel tracing methods, one based on sequence matching and two based on hidden Markov models. Chapter 5 outlines the implementation of the fixation identification and tracing algorithms in a single system, EyeTracer, that allows for protocol manipulation, analysis, and visualization. The following sections summarize these chapters in turn.

Fixation Identification

Fixation identification is the process by which data points that represent fixations are distinguished from those that represent saccades. This process is a crucial step in tracing eye-movements that translates raw eye-movement data into sequences of fixation tuples indicating location, onset time, and duration. While all previous work using eye movements has utilized fixation identification in some way, researchers often have not described the process in detail and have largely ignored the effects of different methods on analysis (with notable exceptions, e.g., Karsh & Breitenbach, 1983). This thesis formally describes both existing and novel methods for fixation identification. In addition, it compares and contrasts these methods in terms of speed, robustness to data noise and variability, and generality to different types of eye-movement data.

Fixation identification methods, broadly speaking, fall into three categories: velocity-based, dispersion-based, and region-based. Velocity-based methods identify fixations according to the velocities between consecutive data points: low-velocity points represent fixations, while high-velocity points represent saccades. The simplest method uses a simple velocity threshold, while a more sophisticated method uses probabilistic velocity distributions encoded in hidden Markov models. Dispersion-based methods identify fixation points as points that are grouped closely together. Such methods often incorporate duration thresholds such that fixation points within a certain dispersion must represent at least some minimal amount of time. Region-based methods identify fixation points as points that fall within a fixed region around some relevant information. In contrast to velocity- and dispersion-based methods, region-based methods require that the user select regions of interest prior to fixation identification.

Tracing

This thesis describes three tracing algorithms: target tracing, fixation tracing, and point tracing. As input, the algorithms take an eye-movement protocol, a cognitive process model, and a set of targets (regions of interest). As output, the algorithms generate a mapping from eye movements to model predictions as well as a measure of "goodness of fit." Each algorithm trades off speed and accuracy in different ways: the fastest provides only moderate accuracy, while the slowest provides very high accuracy. All together, they represent a powerful class of algorithms that allow developers to balance the speed-accuracy tradeoff as necessary for particular applications.

The first algorithm, target tracing, starts with a sequence of fixation locations and maps these locations to their nearest respective targets. The target sequence can optionally be transformed such that consecutive gazes on the same target are collapsed into a single gaze. Target tracing then matches the resulting target sequence to the predicted action sequences of the cognitive model. The sequences are compared using a sequence-distance metric (Card, Moran, & Newell, 1983; Kruskal, 1983) based on the number of inserted and deleted targets from one sequence to the other. The interpretation of the target sequence is simply the predicted model sequence that best matches the target sequence. This interpretation includes a mapping from the observed to the predicted sequence, which helps identify whether or not individual gazes are predicted by the model. Target tracing provides lesser interpretation accuracy compared to the other algorithms but can run significantly faster.

The other two algorithms, fixation tracing and point tracing, use hidden Markov models (HMMs) to interpret eye movements. HMMs are probabilistic finite state machines that have been used effectively in many areas of study, most notably in the fields of speech and handwriting recognition (see Rabiner, 1989). In many ways, the analysis of eye movements has much in common with speech and handwriting recognition. These recognition 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. The tracing algorithms described here perform the analogous task for eye movements, taking a person's eye movements and determining the most likely sequence of intended fixations. Fixation and point tracing borrow a number of techniques from these well-developed fields and carry them over to eye-movement data.

Fixation tracing utilizes HMMs to map fixations onto predicted model sequences. Fixation tracing begins by creating a *tracer HMM* that embodies a grammar representing all possible model sequences. The tracer HMM comprises a number of *submodel HMMs* that represent single fixations on particular targets; the submodel HMMs contain probability distributions for the possible x and y locations of fixations over the target, as well as probability distributions for transitions through and around the submodel. The tracer HMM thus contains a probabilistic representation of possible model sequences and possible fixation locations for each target in those sequences. Fixation tracing then determines the most likely interpretation of a fixation sequence given a tracer HMM; in other words, it maps fixations in the fixation sequence onto states in the HMM. Fixation tracing provides more accurate interpretations than target tracing due to its probabilistic representation: Although it tends to assign fixations to their nearest targets, as does target tracing, fixation tracing has the freedom to guide interpretation to other targets if they are deemed more likely by the given model.

Point tracing operates much like fixation tracing, except that it combines fixation identification and tracing into a single step that maps raw eye-movement data directly to predicted model sequences. Point tracing also forms a tracer HMM, but its submodel HMMs represent properties of data points rather than fixations. Specifically, each submodel HMM includes two states: one that represent high-velocity saccade points, and one that represents low-velocity fixation points. Point tracing then determines the most likely interpretation of a given data point sequence given the model HMM. Point tracing, like fixation tracing, has the freedom to guide interpretation of points to non-nearest targets. However, unlike fixation tracing, point tracing allows the model to influence whether particular data points are identified as fixations or saccades. Point tracing is thus the most powerful of the three algorithms but can incur substantially more computation.

EyeTracer

EyeTracer is an interactive environment that allows for protocol manipulation, analysis, and visualization. Along with all the fixation identification and tracing algorithms mentioned above, EyeTracer includes various ways to visualize eye-movement protocols. Protocol visualization allows for both exploratory and confirmatory analysis. For exploratory purposes, the earlier figures illustrate a type of visualization that incorporates spatial as well as time-sequence information (i.e., coloring from dark to light). EyeTracer allows for other such views that include some minimal processing to assist analysis—for instance, highlighting or numbering fixations. For confirmatory purposes, EyeTracer provides ways of visualizing where protocols match and mismatch with predicted model sequences. For example, EyeTracer can trace protocols with a given model and highlight predicted fixations differently from those which are not predicted.

Applications

To test the fixation identification and tracing methods, the thesis applies the methods to three domains that nicely illustrate their power and flexibility. Chapter 6 describes an application to equation solving that compared the interpretations of the tracing methods to those of human experts and showed how they facilitate the development of cognitive models. Chapter 7 details an application to reading that demonstrated how the tracing methods help evaluate existing cognitive models and how they handle computationally complex models. Chapter 8 describes an application to eye typing that showed how the tracing methods can assist in real-time analysis for use in eye-based user interfaces. The next three sections detail these applications.

Equation Solving

The equation-solving application included two studies that tested the ability of the tracing algorithms to generate correct interpretations and to facilitate model prototyping and refinement. The equation-solving task, as described earlier, involves solving equations of the form $b\ \mathbf{x}\ /\ ac = bd\ /\ a$. The solution is found by encoding the four numeric quantities b, ac, bd, and a and computing the value of $\mathbf{x}$, namely $\mathbf{x} = cd$. There exist a number of possible strategies for finding the solution; for instance, the subject could read the values left-to-right (*b-ac-bd-a*) or right-to-left (*a-bd-ac-b*), or could read the values in pairs (*b-bd-ac-a*, *ac-a-b-bd*). Clearly, a subject's eye movements reveal a great deal about which strategy the subject chooses to execute. However, given eye-tracker noise and individual variability, the mapping from eye movements to subject intentions is often not clear.

Study 1 compared the interpretations of the tracing algorithms to those of expert human coders. In the experiment, subjects were instructed to execute one of four particular solution strategies. These instructions provided a "correct" interpretation of the subjects' eye movements, in that we knew (with at least some reliability) what cognitive processes generated those eye movements. The tracing algorithms and two expert human coders were then given a subset of the protocols and asked to classify them as one of the four strategies. Results demonstrate that the tracing algorithms interpreted the protocols at least as accurately as the human coders in significantly less time.

Study 2 demonstrated how the tracing algorithms facilitate the prototyping and refinement of cognitive process models. In this experiment, subjects solved the equations in whatever manner they chose; thus, there were no longer a priori predictions concerning their cognitive strategies as there were in Study 1. An ACT-R model of subjects' equation-solving behavior was developed using the tracing algorithms for exploratory and confirmatory analysis. First, the tracing algorithms without model bias and the visualization techniques built into EyeTracer helped develop a prototype ACT-R model. Then, the tracing algorithms were applied iteratively to both model and data to evaluate and refine the model. In each iteration, tracing manifested mismatches between subject protocols and model predictions that highlighted weakness of the model and allowed for model refinement. The tracing algorithms, in conjunction with the final ACT-R model, helped reveal relationships between encoding and computation that were hidden in other forms of analysis.

Reading

The reading application tested the ability of the tracing algorithms to compare cognitive models and handle complex models whose strategies cannot be enumerated. The reading task was taken from Schilling, Rayner, and Chumbley (1998) and involved reading medium-length sentences of approximately ten words each. Reichle, Pollatsek, Fisher, and Rayner (1998) fit these data with a computational model of reading, E-Z Reader. They developed several versions of the E-Z Reader model and found that two versions, E-Z Reader 3 and 5, fit the data equally well quantitatively with respect to gaze durations and probabilities. However, they concluded qualitatively that E-Z Reader 5 was the better model because it utilized the fact that words farther in the periphery are more difficult to encode and identify.

The reading study in the thesis compared E-Z Readers 3 and 5 with respect to sequential information. The models were run a number of times to produce first-order and second-order models—that is, models that included first-order and second-order transitions, respectively. These models were used to trace a newly-collected data set and were compared with respect to the goodness-of-fit each model provided for the data set. Results show that, as Reichle et al. suggest, E-Z Reader 5 fit the data significantly better than E-Z Reader 3. Results also show that the resulting traces from all the models helped manifest interesting patterns in the data (gaze durations and probabilities) that are unclear in simpler analyses.

The reading study provided interesting results not only in comparing models, but also in showing how to trace eye movements with complex models with many possible strategies. In the equation-solving study, the process models enumerate the possible sequences of value encodings; for instance, the process model for Study 1 includes each of the four fixation sequences for the four possible strategies given by instruction. For the reading task, however, the process models cannot enumerate all possible fixation sequences; such a model for E-Z Reader, which implements a generally left-to-right reading strategy that can skip or repeat fixations on words, would be computationally infeasible. Thus, the reading study required simpler models (i.e., the first-order and second-order models) that both were computationally feasible and also maintained interesting sequential aspects of the models. The study illustrated that such simpler models can allow for interesting yet efficient tracing of eye movements in complex domains.

Eye Typing

The eye-typing application tested the ability of the tracing algorithms to analyze data as part of an interactive eye-based user interface. The eye-typing interface allows user to type words by looking at letters on an on-screen keyboard, as shown in Figure 1.4 with a sample protocol from a user typing the word *SQUARE*. Existing work (e.g., Jacob, 1993; Stampe & Reingold, 1995) has shown the great potential for such systems to allow hands-free typing, especially for people with physical disabilities (e.g., Hutchinson, White, Martin, Reichert, & Frey, 1989). However, previous systems used a fairly simple scheme to interpret eye movements and to trigger commands: the user had to fixate precisely over a letter for some minimum time, or dwell threshold, for the letter to be typed. Although the systems worked well with this scheme, they required several restrictions—such as large spacing between letters and long dwell thresholds—that facilitated interpretation but also limited their usability and design.

The eye-typing study determined how the tracing algorithms could allow for more freedom in the design and use of eye-based interfaces. The study examined an experimental eye-typing interface that eliminated restrictions necessary in previous systems. The interface traced user eye movements with process models that incorporated possible user behaviors, that is, sequences of letters. Tracing allowed the interface to make use of sequential and other information, such as word and letter frequencies, to help interpret user protocols; for instance, after seeing a fixation on *Q*, the interface would bias its interpretation to favor interpreting the next fixation as *U*. Results of the study showed that, for an interface with minimal restrictions, the tracing algorithms significantly improved the accuracy of eye-movement interpretations over existing methods. In addition, the results showed that users typed faster in the less restrictive interface and improved their performance with even small amounts of practice.

Issues, Recommendations, and Extensions

The final component of the thesis, Chapter 9, describes a number of issues, recommendations, and extensions pertaining to this work. First, the chapter addresses a number of issues raised in the thesis, including how to develop effective cognitive models for tracing, how to improve cognitive models using hidden Markov model training, and how to implement the various algorithms efficiently. Second, the chapter includes recommendations pertaining to the applicability of the algorithms in different contexts—for instance, which algorithms can feasibly operate in real-time as part of an on-line system. Third, the chapter discusses possible future extensions for this work, including incorporation of time information, extension to dynamic task environments, and generalization to other types of data.

Chapter 2
Background

The automated tracing of eye-movement protocols relates to a variety of background work in three broad areas: eye movements, tracing, and Markov models. Work in eye movements has provided a foundation for understanding the physical characteristics of visual-motor control and algorithms for analyzing eye-movement data. Work in tracing has explored techniques for mapping observed data to predicted actions. Work in Markov models, primarily in the fields of speech and handwriting recognition, has developed a rich body of useful Markov-based algorithms for tracing sequential data with respect to a stochastic model. This chapter provides an overview of work in these areas and describes how this work relates to the work in this thesis.

Eye Movements

The wealth of eye-movement studies in past decades has provided solid evidence that eye movements can reveal a great deal about underlying cognitive processes (Just & Carpenter, 1984; Rayner, 1995). Researchers have successfully utilized eye-movement data to study cognition in a variety of domains, a number of which are listed in Table 2.1. Although some have expressed pessimism over relating eye movements and mental processes (e.g., Viviani, 1990; Harris et al., 1988), these numerous studies have undeniably made significant contributions to their fields through the study of eye movements. This section reviews background work in eye-movement with respect to physical characteristics, data collection and analysis, relation to higher-order processing, and relation to other types of data.

Table 2.1: Sample domains for the study of eye movements and representative work.

Domain	Representative Work
reading	Just & Carpenter, 1976, 1980, 1984; Reilly & O'Regan, 1998; Rayner, 1995; Rayner & Morris, 1990
arithmetic	Ohlsson, Ernst, & Rees, 1992; Suppes, 1990; Suppes et al., 1983
word problems	DeCorte, Verschaffel, & Pauwels, 1990; Hegarty, Mayer, & Green, 1992
analogy	Salvucci & Anderson, in preparation
image scanning	Noton & Stark, 1971; Stark & Ellis, 1981; Intraub, 1981
mental rotation	Carpenter & Just, 1978
television viewing	Flagg, 1978
automobile driving	McDowell & Rockwell, 1978
human-computer interfaces	Aaltonen, Hyrskykari, & Räihä, 1998; Ellis et al., 1998

Saccadic Eye Movements

Research in the low-level physical characteristics of eye movements is essential to understanding and analyzing eye-movement data. This thesis focuses on saccadic eye movements, the most common and best understood type of eye movement. Other types of eye movements, such as smooth or vergence eye movements, are beyond the scope of this work. Interested readers can refer to Kowler (1990), Alpern (1962), Fuchs (1971), or Hoffman (1998) for a detailed discussion of saccadic eye movements and overviews of other types of eye movements.

Saccadic eye movements consist of two alternating events: fixations, where the eye locks on a target while the perceptual systems encodes the image; and saccades, where the eye moves ballistically to the next location to fixate. Fixation durations range from approximately 100 ms to over 1 second, depending on the difficulty of the given task (Rayner, 1983). The distribution of fixation durations over particular domains is approximately exponential (Suppes et al., 1983; Harris et al., 1988), with adults exhibiting a shorter average duration than infants in similar tasks (Harris et al., 1988). Fixation location is extremely stable when the head is supported, with a variability of 10-20 minutes of arc, and approximately 1.5 to 3 times greater when the head is unsupported (Kowler, 1990).

Saccades are (primarily) ballistic eye movements during which no information can be encoded (Kowler, 1990). Saccade duration and peak velocity increase with the saccade's magnitude: the average saccade duration is approximately 30 ms for a 5 degree saccade and 2 ms longer for each additional degree; peak velocity can vary from 200-700 degrees/second for saccades of 5-30 degrees (Fuchs, 1971). The visual and cognitive systems can plan sequences of saccades to execute in rapid progression, incurring additional latency before the first saccade for each planned saccade (Zingale & Kowler, 1987). Saccadic parameters do not seem to deteriorate significantly with prolonged scanning activity (Sen & Megaw, 1984).

Eye Movements as Data

Eye movements are extremely informative as a data source for analysis of human cognition. Perhaps the most important reason for their usefulness is that eye movements indicate, generally speaking, the focus of visual attention. Although visual attention can move independently from eye-gaze location, attention always precedes eye movements: attention shifts to the location of the next fixation before the eye executes a saccade to that location (Hoffman, 1998). Thus, at the very least, eye movements provide clues to where the visual attention focuses at approximately the same time. Just and Carpenter (1984) have even posited an "eye-mind" assumption (for reading) that says that processing occurs while information is fixated and the fixation continues until processing is completed. Although this claim has been debated (Viviani, 1990), the link between eye movements and visual attention is clearly strong and provides researchers with a convenient window into a person's thought processes.

Another reason for the popularity of eye movements as data is the fine temporal grain size that they represent. Today's eye trackers can typically sample eye movements at a rate of 50 to 1000 Hz, or once every 1 to 20 ms. This fine grain size allows researchers to analyze human behavior at a level of detail unachievable with more common types of data. This push toward lower levels of detail has arisen in the broader research community; for instance, the ACT-R theory of cognition, which typically modeled actions at the one-second grain size only six years ago (Anderson, 1993), now typically models actions at the 50 ms level. Deeper understanding of cognition requires data that elucidate behavior at deeper levels, making eye movements an excellent choice as a behavioral data source.

The usefulness of eye movement data is further bolstered by the fact that eye-movement data collection, i.e. eye tracking, is generally non-intrusive. Eye-movement studies with both tracking and non-tracking conditions for the same task have shown no significant differences in performance between conditions (Anderson & Douglass, prep). In addition, eye-movement data collection generally requires no subject training or coaching. These benefits, in conjunction with fine temporal grain size, allow researchers to collect data easily even for tasks with short trials of a few seconds or less.

There are a number of interesting similarities and differences between eye movements and verbal (or talk-aloud) protocols, the most common form of protocol data in past research.

While eye movements give only implicit clues to a person's thoughts, verbal protocols represent (or should represent) an explicit version of a person's thoughts. Thus, verbal protocols can more directly indicate the person's high-level goals and problem-solving strategies. However, the verbalization process (as described by Ericsson & Simon, 1984) can be complicated in its own right, often requiring pre-experiment training. Also, verbalization requires explicit cognition to formulate and generate utterances—potentially altering the problem-solving process. Overall, neither eye movements nor verbal protocols come out a clear winner; the choice is highly dependent on the task domain and the goals of the particular study.

Given the usefulness of both verbal protocols and eye movements, several researchers have utilized them together as complementary sources of data (e.g., Hansen, 1991). Typically, these studies focus on verbal protocols to infer high-level strategies and goals, and focus on eye movements to manifest behavior at a finer temporal scale. In an interesting study, Hansen (1991) explored the effects of subjects' eye-movement protocols in eliciting retrospective verbal protocols. Hansen had subjects use a text editor while their eye movements were being recorded. After they completed the task, he showed subjects their own eye movements (as "eye-mark recordings" superimposed on the task screen) and asked them to give verbal retrospections of their behavior. Results showed that the eye-movement protocols assisted in recollection of behavior and induced a significant increase in verbalizations concerning visual encoding.

Other researchers have studied information acquisition using alternatives to eye-movement data. In one study, Payne (1976) placed information on face-down cards and required subjects to flip cars individually to reveal their information. More recently, researchers have employed a type of "poor-man's eye tracker": a computer interface in which concealed information is revealed with a mouse-driven cursor (Lohse & Johnson, 1996; Salvucci & Anderson, 1998). Lohse and Johnson (1996) found that a mouse-driven interface results in slower access times than a similar eye-tracking interface; however, they also found that the mouse-driven interface results in more systematic data that are easier to understand and interpret than comparable eye-movement data.

Eye-Movement Data Analysis

Research in eye-movement data analysis has focused almost exclusively on the problem of fixation identification—reducing raw eye-movement data to sequences of fixations and saccades. The identification techniques fall into several rather rough categories. Some work has focused on finding saccades based on eye-movement velocities between data points, usually incorporating a minimum duration of 15-20 ms (Cabiati et al., 1983; Erkelens & Vogels, 1995; Sen & Megaw, 1984). Other work locates fixations by aggregating consecutive data points into clusters based on distance-from-centroid measures, with duration minimums of 100-200 ms (Karsh & Breitenbach, 1983; Stark & Ellis, 1981; Widdel, 1984). Other existing techniques include digital filters (Tole & Young, 1981) and coding by hand (Harris et al., 1988).

With notable exceptions (e.g., Cabiati et al., 1983; Widdel, 1984), eye-movement data analysis is usually described informally and with few details, although it can have a drastic impact on later data analysis (Karsh & Breitenbach, 1983). Existing techniques also generally rely on rather arbitrary thresholds and other parameters that are often highly dependent on the eye-tracking equipment used. This thesis helps formalize and evaluate a number of existing and novel fixation identification techniques and attempts to make them general enough for different types of eye-tracker data. More importantly, the thesis explores a completely novel line of eye-movement data analysis research: the tracing of eye-movement protocols with cognitive process models.

Tracing

Tracing, a rigorous form of sequential protocol analysis, has become increasingly popular in various fields and under various appellations. Cognitive scientists have employed trace-based protocol analysis to develop and refine cognitive process models (Ritter & Larkin, 1994). Researchers in human-computer interaction have employed tracing, especially sequence comparison techniques, to study the fits of user models (Card, Moran, & Newell, 1983). Builders of intelligent tutoring systems have utilized model tracing (Anderson et al., 1989, 1990, 1995) or tracking (Frederiksen & White, 1990) to determine the user's solution path through a student model of the domain. This section reviews work related to tracing, including more general work in protocol analysis and sequential data analysis.

Methodological Foundations

Newell and Simon (1972) provided arguably the most influential contribution to the methodological foundations of tracing. Their work formalized the notion of the problem space and illustrated how subject protocols could help determine a subject's particular solution path through the space. It also demonstrated how one can test process models by mapping their predictions directly onto the observable actions of human subjects. In the context of tasks involving cryptarithmetic and logic problems, Newell and Simon aligned think-aloud protocols with the predictions of task-specific production system models. To a lesser extent, they also discuss eye-movement data in relation with model predictions (taken from Winikoff, 1967), sometimes to the level of tens of milliseconds. Such detailed analyses have set the tone for this thesis and have demonstrated the potential benefits of low-level analysis for larger sets of protocol data. Of course, there is a tradeoff between the level of detail in the analysis employed and the amount of data on which the analysis is employed, as Newell and Simon state:

> Our investigations of human problem solving alternate between two strategies. One is to examine the behavior of an individual problem solving attempt in as much detail as possible. The aim is to determine whether the information processing theory really describes the fine structure of behavior... The other strategy is to ask more broadly whether the theory fits the facts from a substantial range of situations, trading off breadth for depth. (Newell & Simon, 1972, p. 310)

This thesis is an attempt to push the envelope in both "breadth" and "depth," allowing for the examination of larger and more complex protocols at lower levels of analysis.

Another major step in the evolution of tracing came in the development of process models of human-computer interaction. Card, Moran, and Newell (1983) created a modeling system called GOMS to facilitate understanding of user behavior at a fine-grained level. GOMS represents cognition in four components (from whose initials the acronym is derived): a set of Goals, a set of Operators, a set of Methods for achieving the goals, and a set of Selection rules for choosing among competing methods for goals. In one application to manuscript editing, they traced user actions with GOMS model predictions at the level of individual typing and general reading events. Using these traces, Card, Moran, and Newell evaluated various models by computing a percentage match between predicted and observed sequences with a sequence-distance metric. Their work stressed the need to utilize quantitative measures of a trace's "goodness of fit" along with traditional qualitative analyses. The tracing methods in this thesis also provide such quantitative metrics, and in fact utilize a similar sequence-distance metric for one of the methods.

While Newell and Simon (1972) and Card, Moran, and Newell (1983) emphasized the substantive aspects of their work in their particular domains, Ohlsson (1987) helped to highlight the significant methodological contributions of their work. Ohlsson formalized the methodology, which he called trace analysis, as a three-step process:

1. Construct the subject's problem space.

2. Identify the subject's solution path by making use of the sequential information in the protocol.

3. Hypothesize the subject's strategy by inventing problem-solving heuristics that can reproduce the subject's solution path.

Ohlsson's application of trace analysis to the domain of spatial reasoning nicely illustrates how trace analysis facilitates both exploratory and confirmatory analysis in prototyping and evaluating cognitive process models.,

Ritter (1992) and Ritter and Larkin (1994) examined tracing at length and specify a methodology, trace-based analysis, for testing process models' predictions through comparison with verbal and non-verbal protocols. Their formulation of trace-based protocol analysis, reminiscent of Ohlsson's (1987) trace analysis, comprises the following steps:

1. Using a process model, generate a sequence of predicted actions.
2. Compare the model predictions to empirical data by forming a mapping between the predicted action sequence and the observed action sequence.
3. Analyze the fit of the model to the data to see where the model can be improved.
4. Refine the model and iterate.

Using their Soar/MT system, Ritter and Larkin successfully developed a process model for users of a computer interface. Their paper mentions eye movements only in passing, characterizing them as "overt task actions" that can be handled by the system, but without explaining in detail how this might be achieved. One significant difference between this work and the thesis work herein arises in the predictions of the cognitive model: Soar/MT requires that the model's sequence of predicted actions be deterministic, whereas the tracing methods in the thesis allow for non-deterministic action traces. In general, however, their work is quite applicable in the context of eye movements and maps out a good methodology for developing cognitive models of visual processing.

Ericsson and Simon (1980, 1984) stressed the need for analyzing and tracing protocol data, specifically verbal reports, in the context of the processes that produce them. They emphasize that verbal reports represent observable data that can be collected and analyzed like any other data, given a proper understanding of the process of verbalization. They also outline a model of verbalization and provide numerous suggestions for collecting and analyzing verbal data in the context of this process. Similarly, this thesis attempts to demonstrate the importance of methodology in analyzing eye movements and the need for a rigorous, formal understanding of potential techniques for analysis.

Automated and Semi-Automated Analysis

Because of the tedious nature of tracing protocols, several researchers have attempted to automate the process. For instance, Waterman and Newell (1971, 1973) developed two systems for the automated analysis of verbal reports. Their first system, Protocol Analysis System I (PAS-I), mapped subjects' verbal protocols onto a problem behavior graph, which describes the trajectory of a subject's solution through a problem space (Waterman & Newell, 1971). This system ran without user input and was specifically coded for interpreting protocols in the cryptarithmetic task. The second system, PAS-II, generalized the analysis to any task and allowed the user to interactively take part in the analysis: the user "can provide answers to subproblems the system is unable to solve, correct processing errors, and even maintain control over the processing sequence" (Waterman & Newell, 1973). In a similar effort, Bhaskar and Simon’s (1977) Semi-Automated Protocol Analysis (SAPA) system provided interactive analysis in the domain of thermodynamics.

Although these systems represented a significant attempt at automating protocol analysis, the systems, as the authors themselves admit, constitute only one component task of the larger picture of protocol analysis. A truly automated verbal protocol analysis would allow that we transform language input directly to an interpretation, which would require, among other things, a complete process theory of the generation of verbal reports. Even with important contributions in that area (e.g., Ericsson & Simon, 1984), such a theory remains out of reach. Fortunately with regard to eye movements, we know a great deal about how the cognitive system sends commands to the eye and how the eye executes these commands. This crucial difference between verbal and eye-movement data makes eye movements, in this sense at least, more amenable to automated analysis.

In another attempt to automate tracing, Smith, Smith, and Kuptsas (1993) employed cognitive grammars to represent cognitive strategies and parse verbal, keystroke, video, and action protocols. Utilizing a cognitive theory of writing, they implemented three types of cognitive grammars for an expository writing task: a production rule grammar, an augmented transition network, and an episode grammar. All three grammars could successfully parse subject protocols into a parse tree of higher-order cognitive actions, symbolizing the model’s interpretation of the observed behavior. Although parsing subject protocols is an intriguing method of protocol analysis, and has been effective in the writing domain, its applicability to eye-movement data seems dubious for one primary reason: the parsing must be complete and exact, with no allowance for deviations from the predicted model sequences. The authors claim that to show the validity of some grammar, “the grammar should parse any protocol produced” in a given task. Although we can strive to achieve this ideal, real data, especially eye-movement data, seem simply too noisy for such a strict interpretation. The process of tracing eye-movement protocols must incorporate robust methods that tolerate noise and unexpected or unpredicted actions.

Automated tracing has been employed successfully in intelligent tutoring systems. For instance, Anderson, Corbett, Koedinger, & Pelletier (1995) describe how model tracing can map student actions in a tutoring system to the predictions of an ACT-R (Anderson & Lebiere, 1998) cognitive model. Model tracing in such systems assumes that each student actions corresponds to a unique problem-solving strategy and cannot backtrack in the case of ambiguous actions. It is also limited to the analysis of non-noisy actions such as key presses and mouse clicks.

Several other researchers have investigated automated and semi-automated techniques that do not implement tracing per se but do highlight common goals to the above work and this thesis. Lallement (1998) used decision trees to classify data from an air-traffic controller task. His work showed that such machine learning techniques can provided automated analysis that is more consistent and faster than analysis by hand. Fisher (1987, 1991) constructed the Protocol Analyst's Workbench (PAW) for exploratory data analysis of arbitrary sequential protocol data. PAW provides flexible methods for data encoding, hypothesis testing, and data analysis, with an emphasis on cycle-based analyses such as the determination of so-called Fisher cycles. Sanderson et al.'s (1994) MacSHAPA system allows for similar types of sequential protocol analysis, including sequence comparisons, Fisher cycles, Markov transition statistics, and lag sequential analysis.

Cognitive Process Models

Tracing requires a cognitive process model that can generate predicted action sequences to be matched up against observed action protocols. There are a number of modeling systems that allow for such models, including ACT-R (Anderson & Lebiere, 1998), Soar (Newell, 1990), EPIC (Meyer & Kieras, 1997), 3CAPS (Just & Carpenter, 1992), GOMS (Card, Moran, & Newell, 1983), E-Z Reader (Reichle et al., 1998), and UCIE (Lohse, 1993). In general, the process model used for tracing must be able to produce possible action sequences at the level of the data being modeled; because this thesis deals with visual processing, it assumes that the process model can predict sequences of fixations or gazes. The predicted sequences can be represented as a simple set of sequences or as a grammar that can generate all possible sequences (Smith, Smith, & Kuptsas, 1993).

Limited effort has gone into finding automated methods of generating appropriate process models for a task domain (Langley & Ohlsson, 1989; VanLehn & Garlick, 1987). For a given problem space, these systems infer the conditions under which operators can apply using positive and negative training examples. While the systems do address part of the modeling problem, they still require a full specification the problem space (composed of a representation and operators), which in itself is a major component of the modeling process. However, such efforts suggest that the future of automated modeling seems promising.

Markov Models

One of the primary contributions of this thesis is a class of novel algorithms for tracing eye movements using hidden Markov models. Markov and hidden Markov models have been employed extensively and successfully in the fields of speech recognition (e.g., Rabiner, 1989) and handwriting recognition (e.g., Nathan et al., 1995). In many ways, the application of speech and handwriting techniques to eye movements is quite natural, since they all involve interpreting noisy low-level data in terms of a cognitive or linguistic model. Markov models have also been applied in telerobotics to learn control skills (Yang, Xu, & Chen, 1994) and in biology to align protein sequences (Baldi et al., 1994). However, few researchers of eye movements to date have utilized Markov models, and their work has incorporated these models in only limited ways.

The most common use of Markov models in eye-movement research has appeared in analyses of the transition probabilities from one area of information to another (Schumacher & Korn, 1983; Stark & Ellis, 1981; Suppes, 1990). This work has assumed, explicitly or implicitly, that sequences of fixations are Markovian in character—that is, the probability of a transition from one fixation to another does not depend on the prior sequence of fixations. Though some have argued that eye movements do indeed have this character (e.g., Suppes, 1990), it seems likely that this Markovian assumption simply does not hold in general. However, because the assumption leads to much simpler data analysis, it may be useful for investigating general problem-solving processes in certain contexts.

Only very few researchers have applied the more powerful hidden Markov models to eye movements. Kowler, Martins, and Pavel (1984) used a hidden Markov model to interpret binary observables that represented smooth eye movements in one of two directions. Their model included only two states (one for the expectation of moving in each direction) where the states contained an observation probability for each direction. Rimey and Brown (1991) modeled explicit foveal sequences by means of an augmented hidden Markov model with feedback. The feedback loop allows the model to perform simple cognitive actions, namely directing the eye to the next point of attention, but falls short of the more complete process model approach presented here.

Chapter 3

Fixation Identification

Introduction

The primary objective of eye-movement data analysis, generally speaking, is to determine where, when, and for how long a person looks at targets in the visual scene. Eye-movement data certainly provide this information, but they also provide at least two types of additional information that are extraneous for most analyses. First, the data indicate the paths of each saccade from one fixation location to the next. Because no information can be obtained during a saccade (Fuchs, 1971), the saccade paths say little about a person's encoding behavior. Second, the data (depending on the accuracy of the eye tracker) may include much smaller eye movements that occur during fixations, such as tremors, drifts, and flicks (Alpern, 1962; Ditchburn, 1981). While researchers have debated the usefulness of smaller movements for obtaining information (Ditchburn, 1980; Kowler & Steinman, 1980), they usually mean little in higher-level analyses.

To eliminate these extraneous factors, analysis of saccadic eye movements typically begins with *fixation identification*—the translation of raw eye-movement data points to fixation locations. Fixation identification (or simply identification, for the purposes of this exposition) takes as input an eye-movement protocol comprising a sequence of <x,y> data points. It then transforms the protocol to a sequence of <x,y,t,d> fixation tuples, where x and y indicate the fixation location, t indicates the onset time at which the fixation occurred, and d indicates the duration of the fixation. Identification significantly reduces the size and complexity of the eye-movement protocol, removing saccade points and collapsing fixation points into a single representative tuple.

Though widely employed, fixation identification and its implementation have often been given short shrift in the eye-movement literature, especially literature concerning the interaction of eye movements and higher-level cognition. However, identification is often a critical aspect of eye-movement data analysis that can have significant effects on later analyses; for instance, Karsh & Breitenbach (1983) demonstrated how different identification algorithms can produce vastly different eye-movement traces when analyzing identical protocols. This chapter reviews and formalizes several representative identification methods that researchers have described and successfully employed in previous work. The chapter also includes a novel method for performing identification using hidden Markov models. In conclusion, the chapter summarizes the advantages and disadvantages of the various methods with respect to accuracy, speed, robustness, ease of implementation, and parameter space.

Methods

Velocity-Threshold Identification (I-VT)

Velocity-threshold fixation identification (I-VT) is the simplest of the identification methods to understand and implement. I-VT is a velocity-based method that separates fixation and saccade points based on their point-to-point velocities (Sen & Megaw, 1984; Erkelens & Vogels, 1995). The velocity profiles of saccadic eye movements show essentially two distributions of velocities: low velocities for fixations, and high velocities for saccades. This aspect of saccadic eye movements makes velocity-based discrimination fairly straightforward and robust.

I-VT begins by calculating point-to-point velocities for each point in the protocol. Each velocity is computed as the distance between the current point and the next (or previous) point. I-VT then classifies each point as a fixation or saccade point based on a simple velocity threshold: if the point's velocity is below threshold, it becomes a fixation point, otherwise it becomes a saccade point. The process then collapses consecutive fixation points into fixation groups and discards saccade points. Finally, I-VT translates each fixation group to a $<x,y,t,d>$ fixation tuple using the centroid[1] of the points as x and y, the time of the first point as t, and the duration of the points as d. Pseudocode for the I-VT method is shown in Table 3.1.

I-VT requires the specification of one parameter, the velocity threshold. If angular velocities can be computed (i.e., the distance from eye to visual stimuli is known), the point-to-point velocity threshold can be approximated from a reasonable angular velocity threshold

---

[1] The centroid, or center of mass, of a group of $<x, y>$ points is computed as $<x', y'>$, where x' and y' represent the average of the points' x and y coordinates, respectively.

(Fuchs, 1971); for instance, Sen and Megaw (1984) used a threshold of 20 degrees/second. However, when only point-to-point velocities are known, an appropriate value for the velocity threshold may need to be inferred based on aspects of data collection (e.g., sampling frequency) along with some exploratory data analysis.

Table 3.1: Pseudocode for the I-VT algorithm.

```
I-VT (protocol, velocity threshold)
   calculate point-to-point velocities for each point in the protocol
   label each point below velocity threshold as a fixation point, otherwise
      as a saccade point
   collapse consecutive fixation points into fixation groups, removing
      saccade points
   map each fixation group to a fixation at the centroid of its points
   return fixations
```

I-VT is straightforward to implement and runs very efficiently. However, it sometimes encounters problems when point velocities hover near threshold because of eye-tracker noise or time-averaged data. Specifically, when point velocities are near threshold (e.g., midway between saccades and fixations), the strict threshold can result in "blips" in identification—fixation or saccade groups with only one or a few consecutive points. The protocol in Figure 3.1(a) illustrates this problem: the first real fixation in the protocol is separated into multiple fixations (fixations 3-6), as are the other fixations in the protocol. The problem is especially prevalent when analyzing time-averaged data, such as those in the figure. Researchers have alleviated the problem in the past by aggregating consecutive fixations over a single target into *gazes* on that target (Just & Carpenter, 1984), as shown in Figure 3.1(b). Other work required minimum durations for either fixations or saccades; for instance, Sen and Megaw (1984) required that saccade points must remain above the velocity threshold for a minimum duration of 10 ms.

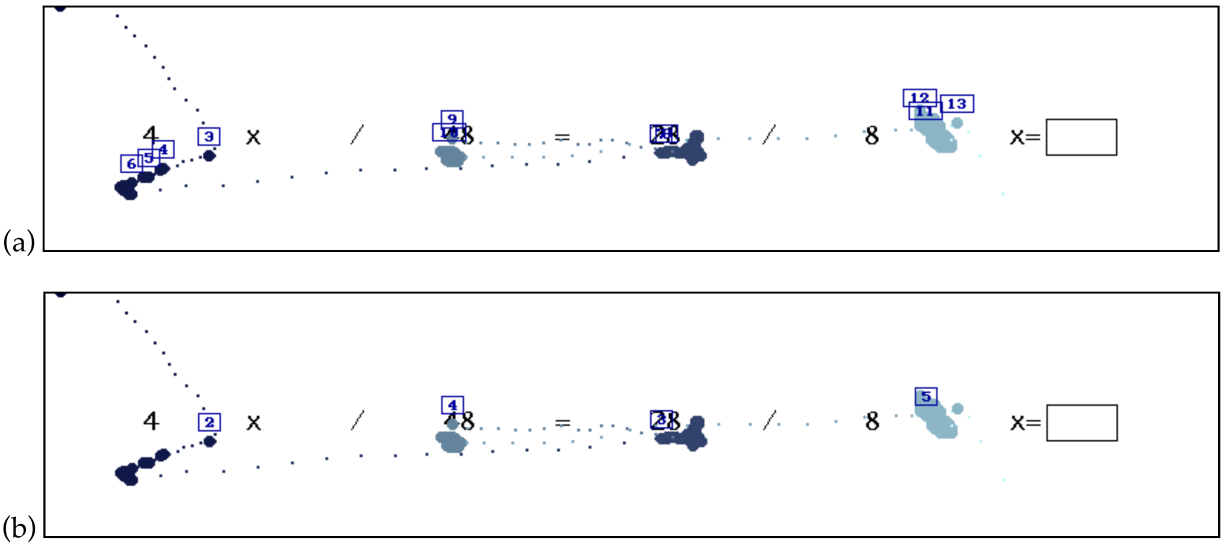


Figure 3.1: I-VT analysis of a sample protocol with (a) fixations and (b) gazes.

Hidden Markov Model Identification (I-HMM)

Hidden Markov model fixation identification (I-HMM) uses probabilistic hidden Markov models to determine the most likely identifications for a given protocol. Hidden Markov models (HMMs) are probabilistic finite state machines that have been employed extensively in the fields of speech and handwriting recognition (see Rabiner, 1989). I-HMM uses a two-state HMM in which the states represent the velocity distributions for saccade and fixation points. This probabilistic representation helps I-HMM perform more robust identification than a fixed-threshold method such as I-VT.

The crux of I-HMM is the two-state HMM shown in Figure 3.2. The HMM includes two sets of probabilities: observation and transition probabilities. The observation probabilities for each state (the small v distributions) represent the distribution of expected velocities in that state. The first state represents saccade points, and thus contains a distribution centered around higher velocities; the second state represents fixation points, and thus contains a distribution centered around lower velocities. The transition probabilities for each state (the arrows exiting the states) represent the likelihood of remaining in the state or making a transition to another state. The transition probabilities in Figure 3.2 show a large likelihood of remaining in each state (.95) and a small likelihood or making a transition (.05). Thus, the HMM provides a probabilistic representation of the observations (i.e., velocities) generated during saccadic eye movements.

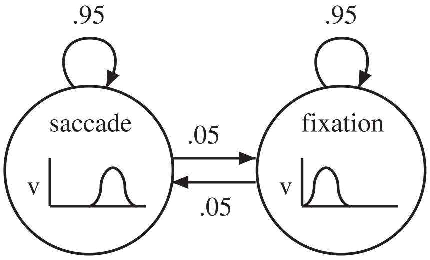


Figure 3.2: Sample two-state HMM. The first state represents higher-velocity saccade points; the second state represents lower-velocity fixation points.

Given the two-state HMM, I-HMM determines the most likely identification of each protocol point through a process of decoding. HMM decoding finds an assignment of protocol points to states that maximizes the probability of the protocol given the HMM, using dynamic programming (Rabiner, 1989) to find this optimal assignment efficiently. (See Appendix A for a full description of HMM decoding.) The assignment associates each point with a state, thus providing an identification of each point as a fixation or saccade point. Like I-VT, I-HMM then collapses consecutive fixation points into groups and outputs fixations at the centroid of the groups. Table 3.2 presents pseudocode for this I-HMM algorithm.

Table 3.2: Pseudocode for the I-HMM algorithm.

```
I-HMM (protocol, HMM)
   calculate point-to-point velocities for each point in the protocol
   decode velocities with two-state HMM to identify points as fixation or
      saccade points
   collapse consecutive fixation points into fixation groups, removing
      saccade points
   map each fixation group to a fixation at the centroid of its points
   return fixations
```

The parameters of I-HMM comprise the observation and probability parameters in the two-state HMM: two observation probability parameters (the mean and variance of the distribution) and two transition probabilities for each state, for a total of eight parameters. While this parameter space is certainly more complex than that of I-VT, the parameters of I-HMM can be learned through a procedure called HMM reestimation (Rabiner, 1989). Given a training set—e.g., a set of protocols—reestimation learns the values of HMM parameters that maximize the probability of the training set given the HMM. (Again, Appendix A includes a full description of HMM reestimation.) Given protocol data from a particular eye-tracking setup, reestimation can provide parameter values that, once estimated, can be used for any protocols collected in that setup.

By employing a probabilistic model rather than a fixed velocity threshold, I-HMM allows for more freedom in identifying points and provides more robust analysis than I-VT, especially in the presence of significant noise. Figure 3.3 illustrates the same protocol as in Figure 3.1 analyzed using I-HMM; the problematic fixations that I-VT separated into multiple fixations (see Figure 3.1(a)) appear as they should in Figure 3.3. Another benefit of I-HMM is that the two-state HMM can be embedded into a larger cognitive model for tracing, as will be discussed in the next chapter. In addition, I-HMM, like I-VT, runs efficiently in linear time.[2] The primary disadvantage of I-HMM is the difficulty of implementing the reestimation procedure, which is both complex and tedious. However, if a public or commercial HMM package is used, or if no reestimation of parameters is needed (e.g., the parameters are provided by other work), the I-HMM algorithm alone (including HMM decoding) is fairly straightforward to implement.

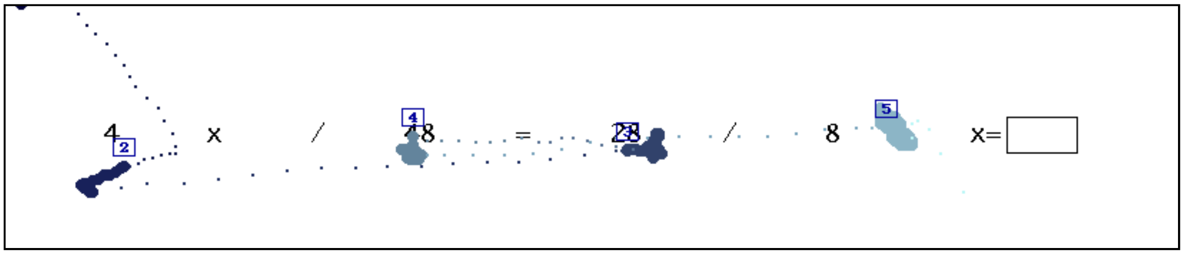


Figure 3.3: I-HMM (and I-DT) analysis of the sample protocol in Figure 3.1.

Dispersion-Threshold Identification (I-DT)

In contrast to the velocity-based identification of I-VT and I-HMM, dispersion-threshold identification (I-DT) utilizes the fact that fixation points, because of their low velocity, tend to cluster closely together. I-DT identifies fixations as groups of consecutive points within a particular dispersion, or maximum separation (Widdel, 1984; Stark & Ellis, 1981). Because fixations typically have a duration of at least 100 ms, dispersion-based identification techniques often incorporate a minimum duration threshold of 100-200 ms (Widdel, 1984) to help alleviate equipment variability. The following I-DT algorithm is based closely on Widdel's (1984) data reduction algorithm.

The I-DT algorithm uses a moving window that spans consecutive data points checking for potential fixations. The moving window begins at the start of the protocol and initially

---

[2] HMM decoding runs in O(N^2T) time, where N is the number of states and T is the length of the protocol being decoded (Rabiner, 1989). For the two-state HMM in I-HMM, N is constant ($N = 2$), thus the algorithm runs in time linear to the length of the protocol.

spans a minimum number of points, determined by the given duration threshold and sampling frequency. I-DT then checks the dispersion of the points in the window by summing the differences between the points' maximum and minimum *x* and *y* values; in other words, dispersion D = (max(x) - min(x)) + (max(y) - min(y)). If the dispersion is above the dispersion threshold, the window does not represent a fixation, and the window moves one point to the right. If the dispersion is below the dispersion threshold, the window represents a fixation. In this case, the window is expanded (to the right) until the window's dispersion is above threshold. The final window is registered as a fixation at the centroid of the window points with the given onset time and duration. This process continues with window moving to the right until the end of the protocol is reached. Table 3.3 includes pseudocode for the I-DT algorithm.

Table 3.3: Pseudocode for the I-DT algorithm.

```
I-DT (protocol, dispersion threshold, duration threshold)
   while there are still points
      initialize window over first points to cover the duration threshold
      if dispersion of window points <= threshold
         add additional points to the window until dispersion > threshold
         note a fixation at the centroid of the window points
         remove window points from points
      else
         remove first point from points
   return fixations
```

The I-DT algorithm requires two parameters, the dispersion threshold and the duration threshold. Like the velocity threshold for I-VT, the dispersion threshold can be set to include 1/2° to 1° of visual angle if the distance from eye to screen is known. Otherwise, the dispersion threshold can be estimated from exploratory analysis of the data. The duration threshold is typically set to a value between 100 and 200 ms (Widdel, 1984), depending on task processing demands.

I-DT is a linear-time algorithm that produces robust identification results. For the sample protocol in Figure 3.1, it generates an almost identical analysis as I-HMM (as shown in Figure 3.3), avoiding the "blip" problem because of its duration threshold. The primary disadvantage of I-DT is the use of two parameters that are highly interdependent; for instance, a small dispersion threshold with a large duration threshold may not result in any identified fixations. Thus, I-DT requires the most careful parameter setting but provides good results for a fairly simple algorithm.

Target-Area Identification (I-TA)

The three previous identification methods can identify fixations at any location in the visual field. In contrast, target-area fixation identification (I-TA) identifies only fixations that occur within specified target areas (DenBuurman, Boersma, & Gerrisen, 1981; Salvucci, Anderson, & Douglass, 1997). The target areas are rectangular regions of interest that represent units of information in the visual field. These target areas, generally used in later analyses like tracing, keep identified fixations close to relevant targets. I-TA also utilizes a duration threshold to help distinguish fixations in target areas from passing saccades in those areas.

I-TA begins by associating data points with target areas: it labels points within a target area as a fixation point for that target, and labels points outside of all target areas as saccades. I-TA then collapses consecutive fixation points for the same target into fixation groups, discarding saccade points. Finally, it removes fixation groups that fall below a given duration threshold and transforms each fixation group into a fixation tuple. The pseudocode for this algorithm appears in Table 3.4.

Table 3.4: Pseudocode for the I-TA algorithm.

```
I-TA (protocol, duration threshold, target areas)
   label each point as a fixation point for the target area in which it
      lies, or as a saccade point if none
   collapse consecutive fixation points for the same target into fixation
      groups, removing saccade points
   remove fixation groups that do not span the minimum duration threshold
   map each fixation group to a fixation at the centroid of its points
   return fixations
```

Although I-TA runs efficiently, it suffers from a somewhat serious problem: the inclusion of many saccade points in identified fixations. Because any point within a target area classifies as a fixation point, saccade points leading into a fixation are often included in the final fixation. This problem can result in deceptively long durations for identified fixations. An even more problematic situation arises when long saccades through large target areas are identified as fixations, especially when the data include some form of time averaging. Figure 3.4 illustrates these problems: fixation 1 includes obvious saccade points, whereas fixation 4 fails to identify fixation points because they lie outside the target areas used. Overall, I-TA can provide reasonable results (e.g., in DenBuurman, Boersma, & Gerrisen, 1981) for certain aggregate analyses but is generally less preferable than the other, more robust algorithms.

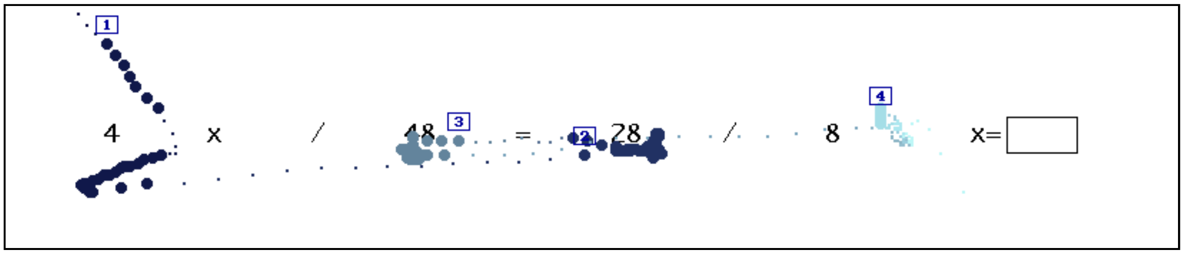


Figure 3.4: I-TA analysis of the sample protocol in Figures 3.1 and 3.3.

Other Methods

The above methods are representative of the most common identification algorithms in previous work. However, other variations do, of course, exist and illustrate the variety of potential techniques. Tole and Young (1981) applied digital filters to perform identification on-line in real-time. Karsh and Breitenbach (1983) utilized a complex algorithm most similar to I-DT in their detailed exposition on identification. Harris et al. (1988) analyzed and coded fixations by hand looking at plots of eye-movement data coordinates over time.

Summary

Table 3.5 summarizes the above identification methods with respect to accuracy, speed, robustness to data noise, ease of implementation, and number of parameters. I-HMM and I-DT provide the most accurate identification, while I-TA performs poorly. All the methods run in linear time, though I-VT has the simplest algorithm as thus the smallest overhead. I-HMM and I-DT give robust results in the presence of eye-tracking noise; I-VT can experience severe “blip” effects when analyzing at the level of fixations rather than gazes. The only difficult method to implement is I-HMM when HMM reestimation is required for parameter estimation. Finally, all the methods require some fine tuning of parameter values.

Table 3.5: Summary of identification methods.

Method	Accuracy	Speed	Robustness	Impl. Ease	Parameters
Velocity Threshold (I-VT)	√	√√	×	√√	1
Hidden Markov Model (I-HMM)	√√	√	√√	√ / × [a]	8 / 0 [a]
Dispersion Threshold (I-DT)	√√	√	√√	√	2
Target Area (I-TA)	×	√	√	√	1+ [b]

[a] I-HMM has 8 parameters in its two-state HMM that can be estimated using the HMM reestimation procedure. If reestimation is used, I-HMM effectively has no parameters but is difficult to implement. If reestimation is not used, I-HMM has 8 parameters but is straightforward to implement.

[b] I-TA has 1 parameter but also requires specification of target areas.

A rigorous comparison of the identification methods requires further context than simply running the methods on various protocols; we must consider how the methods' performance generalizes to different domains and how it affects later types of analysis. Chapters 6-8, which describe the application of the identification and tracing methods to various domains, further examines these issues in the context of the respective domains. However, with our brief glimpse of these methods, I-HMM and I-DT seem to stand out as being more accurate and robust than I-VT and I-TA. Their speed and implementation disadvantages, as we will see, are worth the accuracy and robustness gains and certainly not a serious impediment to their use.

Chapter 4

Tracing

Introduction

Tracing, as outlined in Chapters 1 and 2, entails mapping observed protocols to the sequential predictions of a cognitive process model. Much work has explored uses of tracing for traditional types of data, such as verbal protocols (e.g., Newell & Simon, 1972) or mouse clicks and keystrokes (e.g., Card, Moran, & Newell, 1983). Some of this work has investigated tracing methods that determine the mapping using automated methods (e.g., Waterman & Newell, 1971). However, little work has applied tracing methodology to eye movements, with a few notable exceptions (e.g., Winikoff, 1967). The automated tracing of eye-movement protocols has only been discussed in the context of generic-action tracing methods, and even then has been given merely passing mention (e.g., Ritter, 1992).

This chapter describes three automated methods for tracing eye-movement protocols. The first method, *target tracing*, is an extension of an existing tracing method for generic-action protocols (Card, Moran, & Newell, 1983; Ritter & Larkin, 1994) as applied to eye movements. Target tracing transforms eye movements to generic actions and maps observed to predicted action sequences using a dynamic-programming sequence-matching algorithm. The second and third methods, *fixation tracing* and *point tracing*, are novel algorithms that trace eye movements using hidden Markov models. Fixation and point tracing employ probabilistic models to determine the optimal interpretation for a protocol with a given model. These two methods are designed specifically for eye-movement protocols and thus generate more accurate interpretations than the generic-action algorithm in target tracing.

Tracing takes three inputs: an eye-movement protocol, comprising a sequence of <*x*,*y*> point-of-regard locations as collected by an eye tracker; a set of target areas that designates relevant regions of interest; and a process model, represented as a first-order grammar over target areas. As output, tracing generates a mapping from the protocol to the sequential predictions of the process model and a goodness-of-fit measure that describes how well the protocol fits the model. The remainder of this section further specifies the target area specification and process model representation. The next section describes the individual tracing methods and further specifies the form of the outputs.

Target Area Specification

Target areas, or "regions of interest" (Salvucci, Anderson, & Douglass, 1997), indicate relevant units of information that scanning eye movements are likely to encode. For simplicity, we assume that targets areas can be specified as rectangular regions that enclose the visual space in which the target may be encoded; rectangular regions nicely specify the most common types of information found in visual targets, such as text. In addition, because this thesis is restricted to static task screens, we can assume that the target areas remain fixed throughout a single trial (i.e., the range of one run of the process model).

Let us use the equation-solving task introduced in Chapter 1 as an illustrative domain. In the task, subjects solved equations of the form $b\,\mathbf{x} \,/\, ac = bd \,/\, a$ and mentally computed the value of $\mathbf{x} = cd = (ac/a)(bd/b)$. Figure 4.1 shows one such problem with a sample subject protocol. The figure also includes five rectangles that represent the target areas for this domain, namely, the b, ac, bd, and a values and the *result* box (in which the typed response appears). The size of the target areas indicates approximately the visual space in which each target may be encoded; because the targets are single numbers with large spaces between them, subjects can encode the targets reasonably easily in the parafovea, thus the target areas are fairly large. Such target areas are essential for tracing protocols, allowing the trace to map observed fixations to their intended targets.

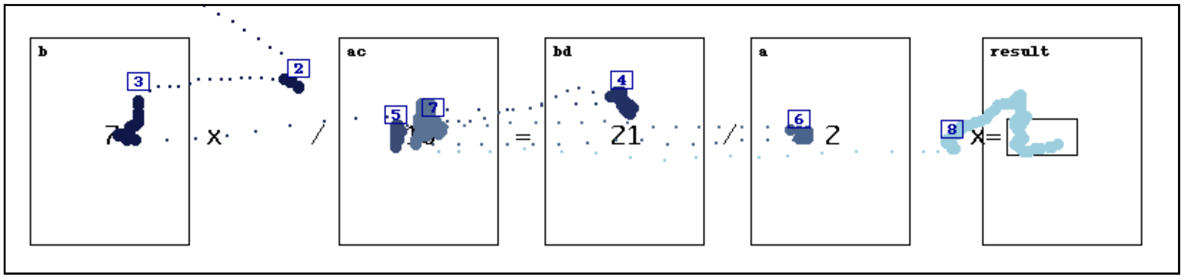


Figure 4.1: Sample equation-solving task screen and protocol with target areas.

An important but subtle note about specifying target areas is that any specification implicitly builds in certain assumptions about behavior in the task. First, the inclusion of a target area signifies that the area is relevant to problem solving. For instance, the omission of target areas for **x**, +, and = in Figure 4.1 signify that these targets are irrelevant, or ignored, in processing; in this case, we have strong reason to believe that subjects do not encode these targets because of the fixed form of equations across problems. Second, the division of target areas relates to the grain size of modeling and data analysis. For example, for a word-problem study of when subjects look at a problem statement, a target area that encloses the entire problem statement may be appropriate. However, for a reading study of word fixation durations, target areas would be required to represent individual words. Third, the layout of the task screen may affect the sizes of the target areas used. For sparse screens with simple information, target areas tend to be large; for dense screens with complex information, targets areas must often be small. Thus, the specification of target areas relies heavily on the goals of data analysis and can significantly affect later analyses. Researchers should take great care to describe the target area specification in expositions involving eye-movement data analysis.

Process Model Representation

The process model provides tracing with a specification of the set of possible predicted action sequences. For the purposes of this thesis, the process model is a *regular grammar* (Sudkamp, 1988, pp. 57-58) that can generate all possible sequences of fixations on targets. The model grammar comprises a set of regular rules built up of *terminals* and *non-terminals*: terminals represent predicted fixations on target areas, and non-terminals represent subgoals that must be achieved through some sequence of fixations. Each rule in the grammar has the form $nt \rightarrow \{t\}^* \{nt\}$, signifying that the left-hand non-terminal can generate a list of terminals followed by an optional non-terminal; in other words, the left-hand subgoal for each rule can generate a sequence of fixations followed by an optional new subgoal.

Let us now develop a process model for the equation-solving domain. Assume that subjects solve equations in one of four strategies, shown as sequences of visual, cognitive, and typing actions in Table 4.1. The first two strategies represent two distinct methods of solving the problem: one in which the values are encoded left-to-right, with intermediate results computed as late as possible; and one in which the values are encoded in pairs, with intermediate results computed as soon as possible. The second two strategies are identical to the first two except that they do not encode (check) the result after typing. Each strategy has a particular probability of occurring based on a .60 probability of reading left-to-right and .40 of reading in pairs, and a .75 probability of checking the result and .25 of not checking.

Table 4.1: Sample equation-solving strategies.

Number	Strategy	Probability
1	encode *b*, encode *ac*, encode *bd*, encode *a*, compute *c*, compute *d*, compute **x**, type **x**, encode *result*	(.60)(.75)
2	encode *b*, encode *bd*, encode *ac*, encode *a*, compute *c*, compute *d*, compute **x**, type **x**, encode *result*	(.40)(.75)
3	encode *b*, encode *ac*, encode *bd*, encode *a*, compute *c*, compute *d*, compute **x**, type **x**	(.60)(.25)
4	encode *b*, encode *bd*, encode *ac*, encode *a*, compute *c*, compute *d*, compute **x**, type **x**	(.40)(.25)

We can represent the strategies in Table 4.1 as the model grammar shown in Table 4.2. This model grammar contains two subgoals, **compute-result** and **type-result**. The **compute-result** subgoal can be executed in one of two ways, represented by the first two rules. The first rule, which executes with probability .60, encodes the values left-to-right; the second rule, with probability .40, encodes the values in pairs. Both rules then execute the **type-result** subgoal. The **type-result** subgoal either generates a fixation on the result using the third rule with probability .75, or generates no fixation using the fourth rule with probability .25. In either case, execution completes due to the absence of a right-hand subgoal for both **type-result** rules.

Table 4.2: Sample process model grammar for the model strategies in Table 4.1.

Number	Rule			Probability
1	**compute-result**	→	*b* *ac* *bd* *a* **type-result**	.60
2	**compute-result**	→	*b* *bd* *ac* *a* **type-result**	.40
3	**type-result**	→	*result*	.75
4	**type-result**	→		.25

A model grammar can either be developed from scratch or derived from any number of existing process and cognitive model representations, such as a production system (Anderson & Lebiere, 1998; Just & Carpenter, 1992; Meyer & Kieras, 1997; Newell, 1990), a user model (Card, Moran, & Newell, 1983; Lohse, 1993), a cognitive grammar (Smith, Smith, & Kuptsas, 1993), or related models (e.g., Reichle et al., 1998). However, there may often be benefits to deriving model grammars from other representations, especially the potential to understand cognitive and motor actions outside the scope of fixations. For instance, the equation-solving grammar in Table 4.2 includes only the fixation sequences for each strategy; it does not include information concerning when values are computed and when the result is typed, as does the strategy set in Table 4.1. By deriving a model grammar from a more sophisticated representation, tracing helps to understand both visual scanning and other behaviors. For example, if a trace indicates that the subject likely read the values in pairs, we have strong reason to believe that the subject computed intermediate values as soon as possible (as the second and fourth strategies dictate) rather than as late as possible (as the first and third strategies dictate). Thus, a trace with respect to the model grammar provides a trace with respect to the original model, preserving all the power of the original model in analysis of the trace.

Process models represented as grammars provide at least three advantages for our goals of tracing eye-movement protocols. First, these grammars allow for an arbitrary hierarchical structure of potential subgoals. Thus, the model can easily represent the embedding of lower-level problem-solving skills into the execution of higher-level skills, as posited by many theories of cognition (e.g., Anderson & Lebiere, 1998). Second, as a related point, the grammars allow for circular execution (i.e., the repetition of subgoals) that are impossible or implausible to represent in a simple set of possible fixation sequences. Third, these grammars translate easily and efficiently to deterministic finite-state automata (Sudkamp, 1988). This aspect of the grammars is critical to the ability to convert model grammars into the hidden Markov models needed for fixation and point tracing.

Tracing Methods

The following exposition describes each of the three tracing methods in order of their sophistication. The descriptions include a specification of the algorithm along with a discussion of implementation issues and general speed and accuracy characteristics. A full evaluation and comparison of the methods is included in the application of the methods to three test domains as described in Chapters 6-8.

Target Tracing

Target tracing is a two-stage process that extends an existing tracing algorithm to eye-movement protocols. Card, Moran, and Newell (1983), Ritter (1992), and Ritter and Larkin (1994) developed tracing algorithms for generic actions based on a common sequence-matching algorithm (Kruskal, 1983). Their algorithms find the optimal match between two sequences that minimizes the *string-edit distance* between the sequences—that is, the number of insertions, deletions, and substitutions needed to transform one sequence to the other. This tracing method, which they applied to user actions and verbal protocols, showed that tracing by sequence matching is highly efficient and generates accurate and useful interpretations for data analysis. When combined with target identification, sequence matching can generate the same useful interpretations for eye-movement data.

Stage 1: Target Identification

The first stage of target tracing transforms the raw eye-movement protocol to a sequence of intended targets. First, target identification performs fixation identification as described in Chapter 3. Then, it assigns each fixation to an intended target, namely the target whose target area lies closest to the fixation. For example, target identification analyzes the protocol in Figure 4.1 to produce the observed target sequence [*ac b bd ac a ac result*]. This method of target identification implicitly makes the assumption that each fixation is intended to encode the visual target nearest to it. In doing so, target identification converts fixations to generic actions as required by the sequence-matching algorithm: a fixation near a target becomes the action of encoding the information in that target.

Stage 2: Target Matching

The second stage of target tracing matches the observed target sequence resulting from the first stage to a predicted sequence of targets generated by the process model. Target matching must first generate all possible target sequences from the process model grammar; for instance, for the grammar in Table 4.2, it generates the sequences [*b ac bd a result*], [*b bd ac a result*], [*b ac bd a*], and [*b bd ac a*]. Next, it utilizes the sequence-matching algorithm to determine the best match between the observed target sequence and each predicted target sequence generated from the model. The sequence-matching algorithm compares the observed and predicted sequences with respect to a sequence-distance metric: the number of insertions, deletions, and substitutions needed to transform one sequence to the other (Kruskal, 1983).[3] (A full description of the sequence-matching algorithm appears in Appendix B.) The predicted sequence with the smallest sequence distance from the observed sequence is chosen as the best-matching sequence. The resulting trace defines a mapping between elements of the observed and predicted sequences. This mapping may contain elements from the observed sequence that are not predicted (i.e., deleted) and elements from the predicted sequence that are not observed (i.e., insertions). In addition, the process generates a goodness-of-fit score as the mismatch between sequences, namely the number of insertions, deletions, and substitutions present in the mapping.

Table 4.3 shows a sample trace for the protocol in Figure 4.1 and the model in Table 4.2. On the left, the table shows the sequence of observed targets in the protocol. On the right, the

---

[3] Some versions of the sequence-matching algorithm do not allow substitutions, but rather treat them as combinations of insertions and deletions. However, qualitative tests showed that, for the purposes of target tracing, sequence matching with substitution assists in handling off-target fixations that are attributed to an incorrect target.

table shows the best-matching sequence of predicted targets from the four possible target sequences described above. This trace shows that protocol most likely represents the strategy in which the equation values are encoded in pairs. In addition, the trace shows that the model does not predict two fixations: the first fixation on *ac*, which is likely a missed saccade to the first element of the equation; and the final fixation on *ac*, which seems to represent a review of previously encoded information. We require two deletions to transform the observed sequence to the predicted sequence, so the mismatch (i.e., sequence distance) between the two sequences is 2.

Table 4.3: Sample trace of the Figure 4.1 protocol using the best-matching predicted target sequence from the model grammar in Table 4.2.

Observed Targets	Mapping	Predicted Targets
ac		
b	→	*b*
bd	→	*bd*
ac	→	*ac*
a	→	*a*
ac		
result	→	*result*

Implementation Issues

The implementation of target tracing is fairly straightforward. Target identification can be implemented using any algorithm in Chapter 3 that suits the particular domain. The sequence-matching algorithm for target matching is described in detail in Appendix B and in several other sources (e.g., Card, Moran, & Newell, 1983; Kruskal, 1983). However, target matching must address two issues that arise specifically in the sequence matching of eye movements. First, the model grammar used in tracing may be circular in that certain subgoals may repeat indefinitely. In such a case, target matching cannot generate all possible grammar sequences, since there exist an infinite number. Thus, when generating grammar sequences, target matching must ensure that circularity is cut off at some convenient point. Second, ties sometimes occur in sequence matching when two or more model sequences have the same sequence distance from the observed sequence. It is unclear whether there is a single best way to break ties in these cases. Qualitative analysis of the equation-solving domain (Chapter 6) seems to indicate that target matching should favor earlier matches over later matches—for instance, favor matches in which the final matched fixation occurs the earliest in the protocol.

However, similar analysis of the eye-typing domain (Chapter 8) indicates that target matching should favor later matches to allow for erroneous fixations that are subsequently corrected. Thus, the method of breaking ties depends on the task domain and cognitive process model. Researchers using and describing target tracing in their work should take care to specify the way in which they deal with these issues.

Speed and Accuracy Characteristics

The efficiency of target tracing depends on the sizes of the protocol being traced and the process model used in tracing. The first stage runs in linear time (see Chapter 3), namely $O(T)$ for a raw protocol of length T. In the second stage, target tracing uses sequence matching between the observed target sequence and each of the M predicted target sequences that can be enumerated from the model. Sequence matching can be done with an efficient dynamic-programming algorithm in $O(F^2)$ time, where F is the length of the longest of the target sequences (Kruskal, 1983). Thus, target tracing requires time $O(T+MF^2)$ to find the best match for a given protocol and process model. Target tracing is especially efficient for smaller process models with few enumerated strategies. Efficiency can degrade with circular process models (depending on the threshold of repetition) and complex hierarchical model grammars with many enumerated strategies.

Target tracing is fairly accurate and efficient, especially for smaller, non-circular process models with few enumerated strategies. However, because of the assignment of fixations to targets in the first stage, target tracing can sometimes have difficulty tracing noisy protocols. Figure 4.2 illustrates such a protocol. When looking at the protocol as a whole, it closely corresponds to the strategy in which values are encoded in pairs, so fixation 2 should be attributed to the target *b*. However, target tracing would assign fixation 2 to its closest target *ac*, resulting in an observed target sequence [*ac bd ac a*] after the first stage. This observed sequence matches equally well with two predicted target sequences, namely [*b ac bd a*] and [*b bd ac a*]. The crucial problem here is the loss of information between stages: once the first stage assigns a fixation to a target, the second stage must rely only on the assigned target for tracing and cannot access information concerning where the fixation may have fallen. A second problem with target tracing involves the lack of ability to utilize rule probabilities in tracing. Although we could imagine heuristics that may help—e.g., multiplying the mismatch score by (1 – strategy probability)—target tracing has no parsimonious and straightforward way of incorporating strategy probabilities.

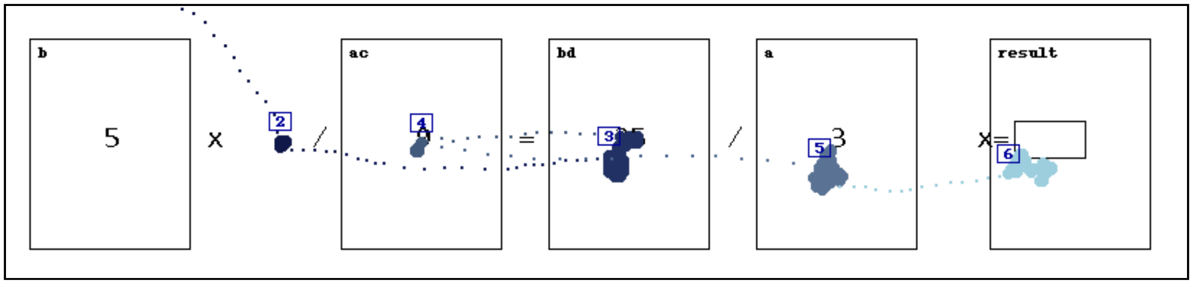


Figure 4.2: Sample equation-solving protocol with ambiguous fixation on *b*.

Fixation Tracing

Fixation tracing, like target tracing, is a two-stage process that comprises identification and tracing. However, instead of converting fixations to targets and tracing the target sequence, fixation tracing keeps fixations as <*x*,*y*> locations and traces this fixation sequence using a hidden Markov model (HMM). Thus, while there is still loss of information (namely the compression of the raw protocol into fixations), fixation tracing uses fixation-location information in the tracing process. This information allows the process to help disambiguate unclear fixations that lie near several possible intended targets.

Stage 1: Fixation Identification

The first stage of fixation tracing identifies fixations as described in Chapter 3. This process results in a sequence of fixations as <*x*,*y*> locations. (Note that the time and duration components of the fixations are discarded; the upcoming section on general issues addresses this point.) Unlike target tracing, fixation tracing makes no assumptions about the intended targets for each fixation in the first stage; this assignment occurs as part of the tracing process in the second stage. The delay of assignment to intended targets allows fixation tracing to facilitate the assignment using higher-level sequential properties of the cognitive process model.

Stage 2: Fixation Decoding

The second stage of fixation tracing decodes, or interprets, the fixation sequence with respect to a *tracer HMM*—an HMM that probabilistically represents the cognitive process model. First, fixation decoding constructs the tracer model from the specified process model and target areas. Second, fixation decoding determines the alignment of the fixation sequence to states of the tracer HMM that maximizes the probability of the fixation sequence with respect to the HMM. We now describe these steps in turn.

The first step in fixation decoding involves building a tracer HMM for the given process model. Because the tracer HMM will decode a fixation sequence, each state in the HMM must represent a fixation. Thus, we begin by constructing *fixation HMMs* for each target area that represent the likely x and y locations for fixations intended to encode that target. Figure 4.3(a) shows a sample fixation HMM for the equation-solving target area b. This fixation HMM contains only one state that includes probability distributions for expected x and y coordinates for fixations intended to encode b; the distributions center around the coordinate centers of the target area, and their standard deviations lie near the boundaries of the target area. The transition probabilities for the HMM force the state to be traversed exactly once. While this HMM is one possible fixation HMM, there are any number of other possibilities, such as those included in Figure 4.3 that allow for (b) repeating and skipping of fixations and (c) unpredicted intervening fixations on any target with uniform x and y distributions. (Fixation HMM (c) is a good choice for many applications.) Overall, the fixation HMM should represent expected fixations around a particular target given that the person intends to look at that target.

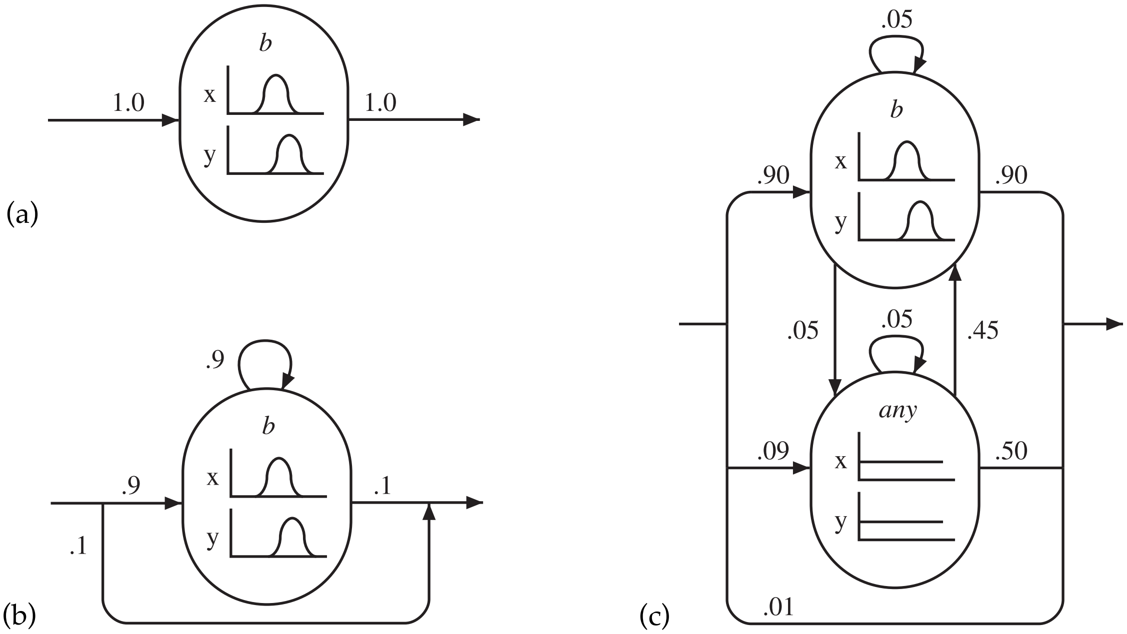


Figure 4.3: Sample fixation-tracing fixation HMMs for target area *b*.

Once the fixation HMMs have been defined, fixation decoding uses the HMMs and the process model grammar to construct the tracer HMM, as shown in Figure 4.4 for the grammar in Table 4.2. Fixation decoding first converts each grammar rule to a *rule HMM* that includes serially-linked fixation HMMs for each predicted target in the rule. For instance, Figure 4.4 contains a rule HMM for rule 1 in Table 4.2 in which the fixation HMMs (from Figure 4.3(c)) for *b*, *ac*, *bd*, and *a* are linked together serially; it also contains a null rule HMM for rule 4. Next, the rule HMMs are linked together according to the non-terminals (subgoals) present in the rules: Each rule HMM with a right-hand-side non-terminal (i.e., rule HMMs 1 and 2) is linked to each other rule HMM with that non-terminal in the left-hand side (i.e., rule HMMs 3 and 4); each rule HMM with no right-hand-side non-terminal (i.e., rule HMMs 3 and 4) is linked to a *terminal HMM* that represents some special *end* observation. The probabilities of the transition links are dictated by the probabilities of the rules to which the transitions go. In the Figure 4.4 tracer model, the rule HMMs for the first and second rules have initial (prior) probabilities that match their rule probabilities. They are linked to the rule HMMs for the third and fourth rules, since the **type-result** subgoal appears in the right-hand side of the former rules and the left-hand side of the latter rules. The probabilities of the transition links reflect the probabilities of the third and fourth rules. The rule HMMs for the third and fourth rules are linked to a terminal HMM that matches the special *end* observation. Note that because the fourth rule HMM is null, the transitions link the first and second rule HMMs directly to the terminal HMM.

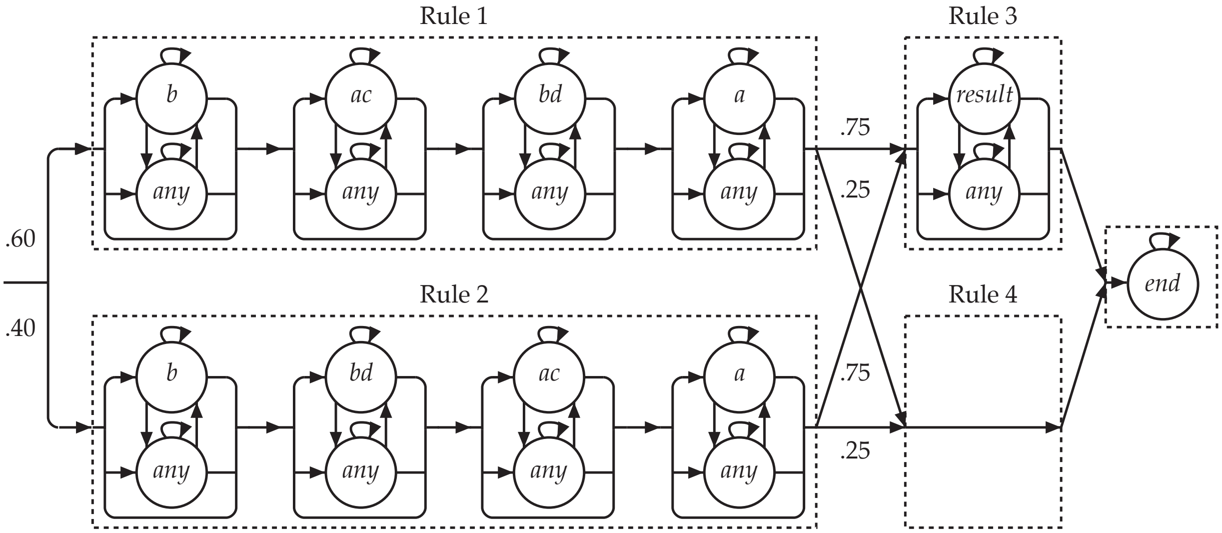


Figure 4.4: Sample tracer model for the model grammar in Table 4.2 using fixation HMMs in Figure 4.3(c).

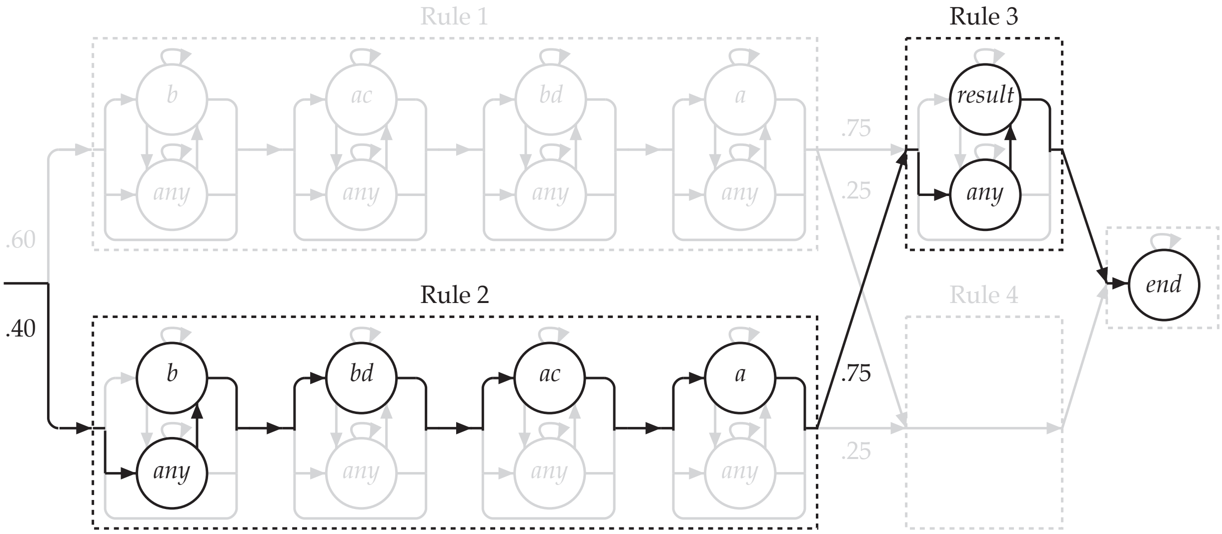


Figure 4.5: Sample trace of the Figure 4.1 protocol given the Figure 4.4 tracer model.

With the tracer model complete, the second step of fixation decoding determines the best alignment of observed fixations to tracer HMM states. This process, called HMM decoding, finds the mapping from fixations to states that maximizes the probability of the fixation sequence given the tracer model. (The HMM decoding algorithm is described at length in Appendix A.) Decoding produces a trace of the protocol in that observed fixations (and thus raw data points) are mapped to their intended targets, as embodied in the tracer HMM states. Figure 4.5 illustrates how the fixations in the Figure 4.1 protocol are mapped to their respective states. The mismatch score produced by fixation tracing is based on the probability $\Pr(O,S \mid \lambda)$ as described in Appendix A. For an observed fixation sequence O, a decoded state sequence S, and a tracer model λ, the mismatch score can be defined as:

$$mismatch = -\frac{\log \Pr(O,S \mid \lambda)}{\text{length}(O)} \qquad \text{(Equation 4.1)}$$

Here the log probability is divided by the number of fixations to represent the log of the average probability per fixation. The negation of this result produces a mismatch score of 0 or greater.

Implementation Issues

The primary difficulty of implementing fixation tracing comes in the construction and decoding of the tracer HMM. Construction of the HMM requires routines to compose smaller HMMs into larger ones, as described above, for linking fixation HMMs into rule HMMs and rule HMMs into the tracer HMM. Decoding with the HMM requires routines to find the optimal match for a given observed protocol, as described above and in Appendix A. A standard public or commercial package for manipulating HMMs greatly lessens the burden of implementing fixation tracing.

Two important issues arise in HMM implementation and use. The first issue is the choice between "dense" and "sparse" HMM optimization. Dense HMMs have very high connectivity between states (e.g., all states are connected to one another), while sparse HMMs have very low connectivity (e.g., each state is connected to only a few others). An implementation of HMMs is typically optimized for either dense or sparse HMMs to make that particular type the most efficient. The choice between dense and sparse HMM optimization is, not surprisingly, dependent on the task domain. However, the cognitive process models tested in this thesis have generally sparse representations, thus it seems that sparse HMMs may be preferable for most domain applications. The second issue is the incorporation of rule tags into tracer HMM states. It is convenient when building the tracer HMM to include within each state not only the name of the target area it represents, but also the rule it represents as indicated by a rule tag—e.g., the rule's number or special name. The rule tags can be examined after tracing to determine the strategy followed in the protocol. For instance, in Figure 4.5, the fact that the trace traverses the rule HMMs for rules 2 and 3 indicate that the subject read the equation values in pairs and subsequently read (checked) the typed result.

Speed and Accuracy Characteristics

Like target tracing, the speed of fixation tracing depends on the number of fixations and the size and complexity of the process model. Again, fixation identification requires $O(T)$ time for a raw protocol of length T. Fixation decoding requires $O(N^2F)$ time, where N is the number of states in the tracer model and F is the length of the observed fixation sequence (Rabiner, 1989). Thus, fixation tracing altogether requires $O(T+N^2F)$ time. This time analysis can be deceiving in two ways. First, fixation tracing can utilize beam decoding (as described in Appendix A) to trade off the optimal standard decoding for improved efficiency. Second, for sparse tracer HMMs, fixation decoding acts more like an $O(T+NF)$ algorithm, since each of the N states is connected to few other states. Third, fixation tracing requires construction of the tracer HMM; however, an implementation should memoize tracer HMMs such that they need only be constructed once for a particular model. Fixation tracing is thus difficult to analyze off-line in the absence of a concrete application context.

Fixation tracing, because of the incorporation of fixation location in the tracing process, can allow for more flexible and accurate interpretations than target tracing. In the Figure 4.2 protocol, while target tracing assigns fixation 2 to target *ac* before tracing, fixation tracing makes use of the fixation's location and the overall sequence of fixations to attribute the fixation accurately to the target *b*. By using hidden Markov models for tracing, fixation tracing is able to employ higher-level global information, namely the sequence of fixations as predicted by the process model, to influence the interpretation of lower-level aspects of the protocol, namely the assignment of fixations to targets. It also makes use of grammar rule probabilities, unlike target tracing. Nevertheless, the two-stage process of fixation tracing still results in some loss of information between stages, namely the determination of where fixations occur in the protocol. Figure 4.6 shows a protocol in the fixation identification stage fails to identify an evident fixation on target *a* (at the direction reversal on the right side of the protocol), most likely because of the fixation's short duration and the effects of time-averaged sampling. This example illustrates how the cognitive model in the second stage cannot influence the determination of what is and is not a fixation.

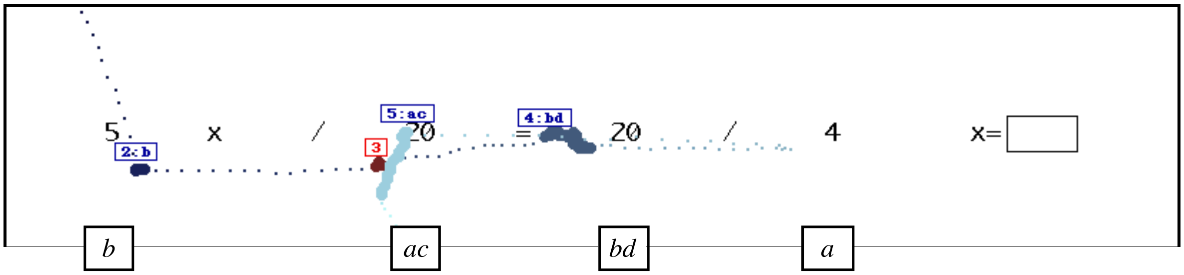


Figure 4.6: Sample protocol with unidentified fixation on target area *a*.

Point Tracing

Point tracing identifies fixations and traces the protocol in a single stage, thus avoiding the loss of information between stages suffered by target and fixation tracing. The point-tracing algorithm relates very closely to the fixation-tracing algorithm: it constructs an HMM from the cognitive process model and decodes the protocol with this HMM. However, rather than decoding a sequence of fixation locations, point tracing decodes the raw sequence of eye-movement data. The one-stage process allows the cognitive process model to influence both the identification of fixations and the assignment of fixations to intended targets.

Point Tracing

Point tracing begins with the construction of a tracer HMM used to decode the given protocol. This process is, for the most part, identical to that in the second stage of fixation tracing: it generates fixation HMMs for each possible target, builds rule HMMs for each rule in the model grammar, and links the rule HMMs as described by the model grammar. However, while the fixation HMMs in fixation tracing represent expected fixation locations, the fixation HMMs in point tracing represent expected $<x,y,v>$ eye-movement data. Figure 4.7 shows three possible fixation HMMs for target *b* analogous to those for fixation tracing in Figure 4.3. Figure 4.7(a) shows a fixation HMM that represents a single fixation on *b*. The first state represents the saccade points present before the fixation; the state includes uniform x and y distributions and a v velocity distribution centered around high velocities. The velocity distributions can be estimated from existing protocol data, as described for the I-HMM algorithm in Chapter 3. The second state represents fixation points with x and y distributions centered around the b target and a v distribution centered around low velocities. The other fixation HMMs in Figure 4.7 represent (b) zero, one, or multiple fixations, and (c) zero, one, or multiple fixations with possible intervening fixations. In some sense, these fixation HMMs are a combination of the saccade-fixation HMM in Figure 3.2 and the fixation HMMs in Figure 4.3: they include both location distributions for fixation assignment and velocity distributions for fixation identification.

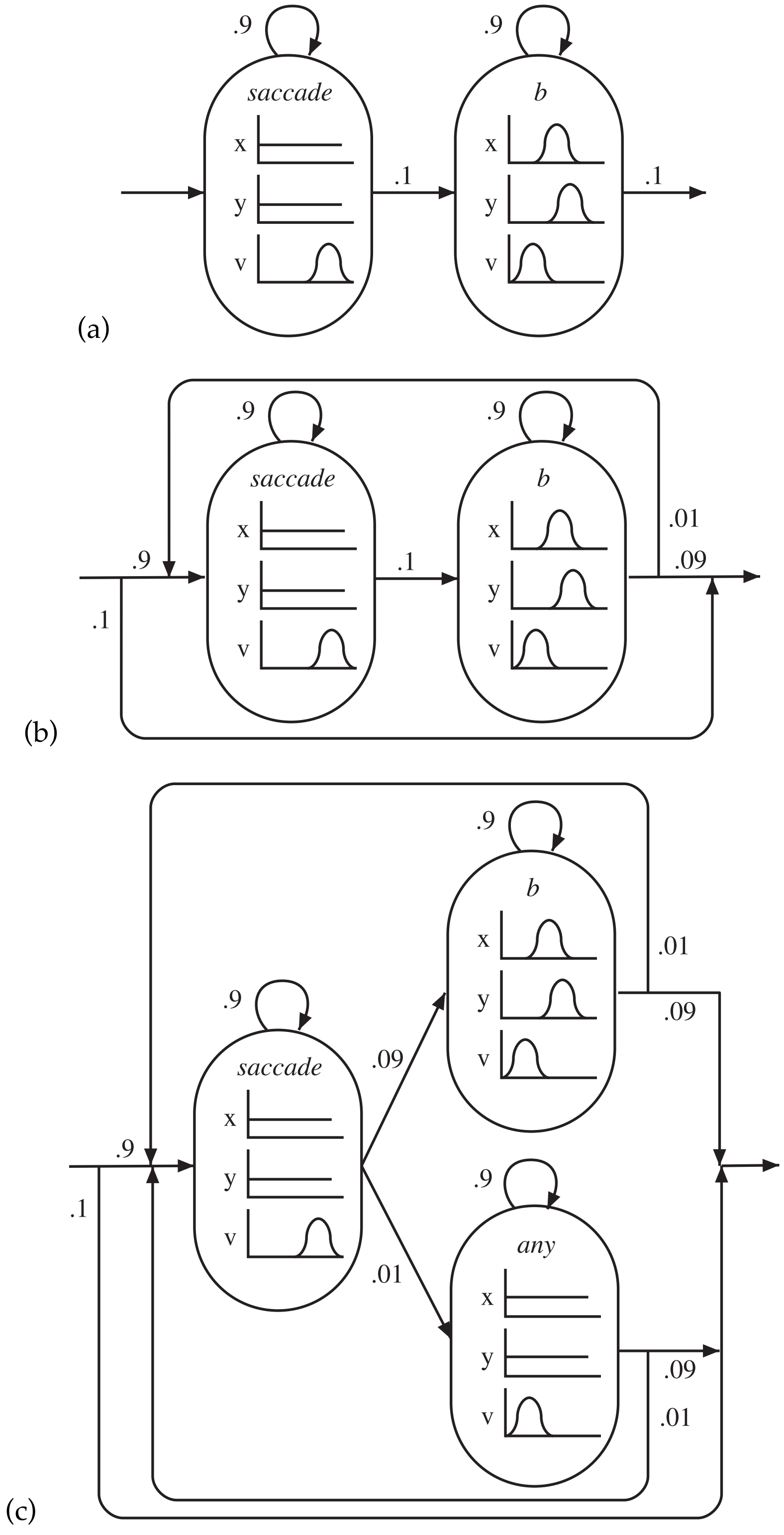


Figure 4.7: Sample point-tracing fixation HMMs for target area *b*.

Using the constructed tracer model, point tracing decodes the given protocol and determines the optimal protocol trace. First, it augments the <*x*,*y*> eye-movement protocol with point-to-point velocities to generate sequences of <*x*,*y*,*v*> tuples. Second, it decodes these tuples with the tracer HMM to find the optimal trace, or mapping, from observed to predicted fixations. The trace provides a one-to-one mapping between data points and tracer model states, which can be used to collapse the raw protocol into fixations or gazes. The mismatch score for point tracing can be defined exactly as that for fixation tracing, as shown in Equation 4.1. Note that the mismatch scores between fixation and point tracing are not directly comparable, since the mismatch for point tracing represents the negative log of the average probability per raw data point rather than fixation.

Implementation Issues

The implementation of point tracing brings up many of the same issues as that of fixation tracing. In point tracing, these issues can be substantially more critical, in that point tracing is algorithmically much less efficient than fixation tracing, as discussed below. Specification of the fixation HMMs can also be more difficult because of the integration of location and velocity information in the HMM states. Again, researchers can utilize a public or commercial HMM package to alleviate the burdens of the HMM implementation.

Speed and Accuracy Characteristics

The one-stage point-tracing algorithm essentially performs fixation identification and tracing simultaneously. Point tracing requires $O(N^2T)$ time for a tracer model with N states and a raw protocol of length T. The number of states N for point tracing typically differs from N for fixation tracing by a factor of 2 or 3, due to the increased number of states in the fixation HMMs. Also, T is often much larger than F in target and fixation tracing, since the latter methods discard a great deal of the information in the raw protocol. Again, beam decoding, sparse tracer HMMs, and tracer HMM memoization and can significantly improve the efficiency of the algorithm. Nevertheless, point tracing is by far the least efficient of the tracing methods.

The loss in efficiency of point tracing is somewhat counterbalanced by gains in information from tracing a raw protocol directly. Unlike fixation tracing, point tracing allows the cognitive model to influence what data points are identified as fixations. For instance, Figure 4.8 shows the same protocol as Figure 4.6 traced with point tracing; here the model determines that the bend in data points nearest the *a* target must correspond to a fixation, even though the point velocities fail to indicate a fixation. Unfortunately, point tracing occasionally fails to identify incidental or extraneous fixations as we would expect: rather than assigning them to some *any* state (see Figures 4.5 and 4.7), point tracing tends to identify points for incidental fixations as saccade points rather than fixation points. The protocol in Figure 4.8 illustrates this problem in missing the incidental fixation on *ac* between 2 and 3 visible in Figure

4.6. This problem causes difficulty when analysis requires the determination of incidental fixations, as will arise in the eye-typing study in Chapter 8.

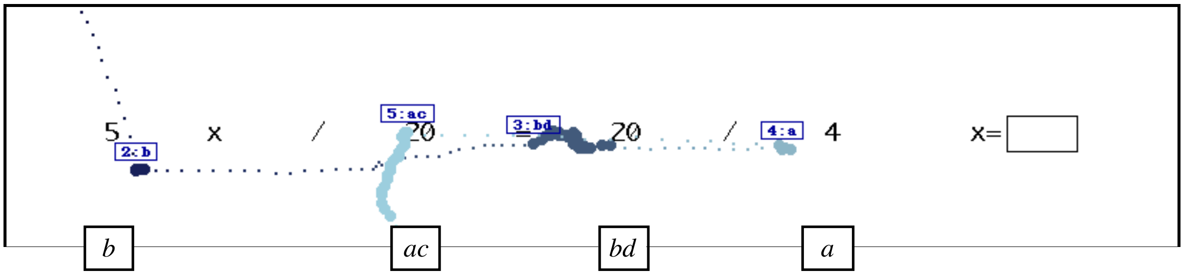


Figure 4.8: Protocol in Figure 4.6 traced with point tracing.

Issues

Using Fixations versus Gazes

Fixations represent a fairly natural grain size for typical analysis and modeling of eye movements. However, for some applications or domains, fixation information may be too detailed for higher-level analysis interested in more general aspects of visual scanning. For such purposes, many researchers, especially in the reading domain, have utilized *gazes* as the basic units of eye-movement analysis (e.g., Just & Carpenter, 1984; Rayner & Morris, 1990). A gaze is typically defined as a sequence of one or more consecutive fixations on the same target area. Gazes allow researchers to focus their analyses on eye movements between visual targets and to ignore eye movements within targets. For instance, Salvucci, Anderson, & Douglass (1997) analyzed eye movements in a simple physics problem-solving task where the problem statement comprised several short sentences. Because they were primarily interested in the visual interplay between the problem statement and an example, they utilized gazes to abstract over information that was extraneous to this analysis—for instance, individual fixations exhibited when reading the problem statement were converted to a single gaze on the entire problem statement area.

All three tracing methods allow a user to trace eye movements at the level of either fixations or gazes. The decision to use fixations or gazes impacts the specification of the cognitive process model and the tracing algorithms themselves. First, the cognitive process model should predict scanning behavior at the level of the desired analysis: If we are interested in tracing gazes in a physics task, our model should predict individual gazes; if we are interested in tracing fixations in a reading task, our model should predict fixations. Second, the tracing algorithms as described generally assume analysis at the fixation level. At the gaze level, the algorithms require slight modification. For target tracing, target identification must collapse consecutive, repeated gazes on target areas into single gazes before passing the target sequence to target tracing. For fixation and point tracing, the fixation HMMs should allow for repeated fixations on their respective targets to model the multiple fixations that can comprise a gaze. The domain studies in Chapters 6-8 illustrate how the tracing methods handle fixations (Chapter 6) and gazes (Chapters 7-8).

Using Overlapping Target Areas

In the specification of target areas, it is sometimes convenient to define overlapping or embedded target areas. Overlapping target areas arise in task screens where fixations on certain screen locations could plausibly represent encodings of multiple targets. For instance, we might define overlapping value target areas for the equation-solving task because of the large spacing between values that facilitates parafoveal encoding, as shown in Figure 4.9. Embedded target areas arise in screens with an implied hierarchy of visual targets. For the equation-solving example, we may wish to define an area *equation* that represents fixations that encode the entire equation—say, during visual search of the task screen. Such target areas may be utilized in a model that predicts an initial fixation to locate the equation and subsequent fixations that encode the individual components of the equation.

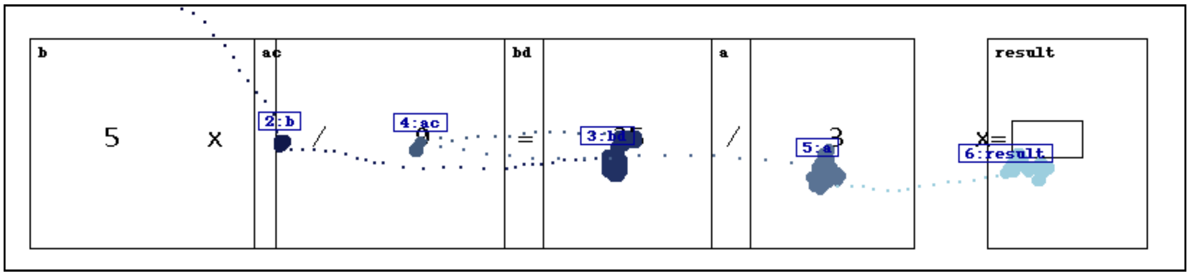


Figure 4.9: Protocol with overlapping target areas.

While fixation and point tracing can handle overlapping and embedded areas, target tracing is not well-suited for such areas. Target tracing maps fixations to the nearest target, thus the mapping of a fixation inside multiple target areas is ambiguous; typically the mapping defaults to whatever area comes first in the area list. While the definition of "nearest target" may be coaxed to into providing a temporary fix for certain cases, target tracing is inherently limited by its lack of the cognitive model influencing the fixation-to-target mapping. Fixation and point tracing, however, do allow the model to influence this mapping through their probabilistic representations of the target areas. Fixations that fall within multiple areas can be assigned to either area, as dictated by the location of the fixation within the distributions and by the sequential predictions of the process model. Figure 4.9 illustrates that a fixation can even lie in one target area but be assigned to another: fixation 2 lies solely within target area *ac* but, because of the process model, is assigned to target *b*. Similarly, when a target areas lies wholly embedded within another area, both areas compete for assignment of a fixation. In this case, the inner target area has a narrow distribution that captures nearby points, and the outer target area has a flatter distribution that captures father points. Again, the process model can heavily influence the competition, such that the model can significantly favor one over the other based on likely strategies represented by the model's predicted fixation sequences.

Using the "Any" Target Area

Some process models of visual attention may not predict the exact target of every fixation, but rather may posit that a fixation lands in some area on "any" target in that area. For instance, in the equation-solving domain, subjects sometimes move their eyes down onto the equation from the start target area to orient themselves as to where elements of the equation lie; the fixation is not intended for a particular target value, but more so for the entire equation as a whole. While such fixations could occasionally be modeled with larger target areas, it is often convenient to utilize a special *any* target area that captures fixations on any target. The *any* target area allows process models to predict fixations not intended for specific targets or incidental to problem solving.

All three tracing methods can handle the *any* target area. We have already seen how fixation and point tracing do this: they utilize fixation HMM states in which the x and y distributions area fairly uniform (flat) and thus do not center over a particular target. Target tracing handles the *any* target area with a slight modification of the sequence-matching algorithm. This modification allows fixations on arbitrary targets to perfectly match the *any* target with no mismatch penalty. The modification is implemented in the sequence-matching algorithm described in Appendix B.

Using Duration and Time Information

The tracing methods described in this thesis utilize the locations of fixations and the sequence in which they occur. Additional information present in an eye-movement protocol, namely fixation durations and onset times, can potentially facilitate interpretation of the protocol. Fixation durations can indicate the amount of cognitive processing dedicated to particular fixations or gazes. This information could help determine whether a fixation is intended or incidental, or whether computation takes place during a fixation; for instance, fixations in the equation-solving task during which the person performs a mathematical operation are typically longer than other fixations (see Chapter 6). Fixation onset times can help to understand the interleaving of encoding and other actions, such as typing or mouse movements. By comparing the onset times of fixations with the times of these other types of actions, tracing methods could potentially distinguish between various strategies with similar eye movements but dissimilar behavior in these actions. The use of duration and onset time information in tracing can be quite complex and is beyond the scope of this thesis. The final chapter explores this future direction in the context of multimodal tracing and tracing for dynamic task environments.

Summary

Table 4.4 provides a summary of the tracing methods with respect to model influences and complexity. The model influences illustrate the power of the various methods in terms of what aspects of the tracing process are influenced by the specified process model. All the methods depend on the length of the raw protocol T. The dependency of target tracing on M is closely related to the dependency of fixation and point tracing on N. For simpler models, M (the number of enumerated strategies) and N (the number of states) may be near equal, but N may be significantly smaller than M for more complex, hierarchical process models. Also, the factor of $O(N^2)$ in fixation and point tracing typically behaves more like $O(N)$ for process models that generate sparse tracer HMMs.

Table 4.4: Summary of the tracing methods.

Method	Model Influences	Complexity [a]
Target Tracing	trace	$O(T+MF^2)$
Fixation Tracing	fixation assignment, trace	$O(T+N^2F)$
Point Tracing	fixation identification, fixation assignment, trace	$O(N^2T)$

[a] The complexity parameters are interpreted as follows: T is the length of the raw protocol, M is the number of enumerated strategies for the process model, F is the number of fixations, and N is the number of states in the tracer HMM.

The tracing methods, like the fixation identification methods, are difficult to compare in the absence of an application domain. Chapters 6-8 provide detailed quantitative and qualitative evaluations and comparisons of the tracing methods and how they interact with the various fixation identification methods. Chapter 9 includes an overview of these results and a number of recommendations for choosing an appropriate tracing and fixation identification method for other application domains.

Chapter 5

EyeTracer

Introduction

EyeTracer is an analysis environment that allows for manipulation, visualization, and post-processing of eye-movement protocols. It embodies all the fixation identification and tracing algorithms described in Chapters 3-4, and also includes variants on these algorithms for comparison and testing. EyeTracer provides various ways to visualize eye-movement protocols to facilitate data analysis. In addition, the system is modular such that for any new task domain, a user can implement only the aspects of the algorithms specific to that particular domain. Thus, EyeTracer provides an excellent testbed for applying the algorithms to the task domains detailed in Chapters 6-8.

EyeTracer is the first system designed specifically for the automated tracing of eye-movement protocols. However, EyeTracer resembles several past and present protocol analysis systems, including MacSHAPA (Sanderson et al., 1994), PAS-I and PAS-II (Waterman & Newell, 1971, 1973), PAW (Fisher, 1987, 1991), SAPA (Bhaskar & Simon, 1977), and Soar/MT (Ritter & Larkin, 1994). Some systems (e.g., MacSHAPA, PAW) emphasize exploratory analysis of protocols with the goals of better understanding behavior and forming an initial cognitive model of this behavior. Other systems (e.g., PAS, Soar/MT) emphasize confirmatory analysis with the goals of comparing observed behavior to model predictions and thereby refining the model. Critical to both types of analysis is the ability to visualize protocols in some convenient form; for instance, Soar/MT provides a spreadsheet in which verbal protocols can be visually aligned with model predictions to make the trace explicit. EyeTracer provides a number of ways to visualize protocols for both exploratory and confirmatory analysis, thus enabling a user to better understand behavior, construct a prototype model, and subsequently refine the model based on observed-predicted mismatches.

EyeTracer runs on the Macintosh platform and is implemented in Common Lisp in the MCL environment. The remainder of this chapter describes how to set up and use EyeTracer for eye-movement protocol analysis. The exposition aims to facilitate understanding of the system for later chapters and to assist in the operation of the system by new users.

EyeTracer Setup

To run EyeTracer, a user must first set up the system by creating an eye-movement protocol data set and by defining aspects of the experiment in which the data set was collected. The creation of a protocol data set requires transforming existing eye-movement protocols into the standard EyeTracer protocol format. EyeTracer protocol format includes two Lisp data structures, `porfile` and `portrial`, that define protocol files and trials, respectively. (The `por` prefix indicates that the structures hold point-of-regard, i.e. eye movement, information.) The `porfile` structure includes the following fields:

- `name`: subject name
- `day`: day (or session) that the file represents
- `condition`: condition under which the subject was run for this day
- `portrials`: trial protocols as portrial structures

Each `portrial` structure has the following components:

- `number`: trial number
- `stimulus`: arbitrary object representing the trial stimulus
- `response`: arbitrary object representing the subject response for the trial
- `pordata`: list of <*x*,*y*> eye-movement data points

The `stimulus` and `response` fields can hold arbitrary Lisp code to define the respective aspects of the trial; the stimulus is required for drawing the task screen, whereas the response is maintained only for later analyses (i.e., after post-processing). EyeTracer assumes that there is a single `porfile` for each unique subject and day.

The second component of setup, defining aspects particular to the experiment, requires the creation of four Lisp configuration files: *Main*, *Stimulus*, *Areas*, and *Grammar*. The files are as follows:

- *Main* : This file defines general characteristics of the experiment and of data collection. Characteristics of the experiment include the experiment name, subjects, and days (or sessions). Characteristics of data collection comprise the type of eye-tracking equipment used, the sample rate, and possibly the type of time averaging used. Table 5.1 shows a sample *Main* file for the equation-solving experiment. The call to `set-dataset` defines the experiment name and subjects, along with indicating that the experiment lasted only one day. The call to `set-datatype` dictates that an IScan eye tracker is being used at a sampling rate of 120 Hz with standard time averaging. EyeTracer has a number of similar built-in data types from which users can select; this specification sets various tracing parameters, such as velocity distributions for I-HMM and point tracing.

Table 5.1: Sample *Main* file.

```
(set-dataset
 "Equation"
 '("Subject1" "Subject2" "Subject3")
 '(1))
(set-datatype "IScan, 120 Hz, Standard Averaging")
```

- *Stimulus* : This file defines the experiment stimuli and how the stimuli are presented on the screen. The file contains the function dataset-`draw-screen` that draws a given portrial to the screen using the information in its `stimulus` field. Because EyeTracer assumes that the task screen is static, it calls the function only once before drawing a protocol over the screen. Table 5.2 shows a sample *Stimulus* file, in Lisp with some pseudocode, for the equation-solving experiment.

Table 5.2: Sample *Stimulus* file.

```
(defun dataset-draw-screen (view portrial)
  <clear view>
  (let ((stimulus (portrial-stimulus portrial)))
    <draw equation in view as described in stimulus>))
```

- *Areas* : This file defines the target areas for the experimental task screen as described in Chapter 4. The function dataset-`portrial-areas` returns a list of target areas for the given `portrial`; note that this function allows different trials, and therefore different stimuli, to include different target areas. Each target area is specified as a list (*name x1 y1 x2 y2*), where *name* is the name of the target area, *x1* and *y1* define the upper-left corner of the area rectangle, and *x2* and *y2* define the lower-right corner of the rectangle. The target areas can overlap to any degree, although overlap may affect certain types of post-processing (e.g., target tracing, as discussed in Chapter 4). Table 5.3 illustrates an *Areas* file that defines the five relevant equation-solving target areas used in previous examples.

Table 5.3: Sample *Areas* file.

```
(defun dataset-portrial-areas (portrial)
  (declare (ignore portrial))
  '((b 67 202 189 346) (ac 333 202 405 346)
    (bd 477 202 549 346) (a 621 202 693 346)
    (result 783 202 927 346)))
```

- *Grammar* : This file defines the cognitive process model grammar. The function dataset-`portrial-grammar` returns a list of grammar rules for the given `portrial`; as for target areas, the function allows different trials to have different model grammars. Each grammar rule comprises a non-terminal, terminal-list, non-terminal, and probability to mimic the rule form nt → {t}* {nt} discussed in Chapter 4. The grammar is built up by passing these elements to the `create-rule` function and combining the resulting rules with the `create-grammar` function. The grammar may be written directly or derived from a different process model representation, such as a production system. Table 5.4 shows a sample *Grammar* file that encodes the equation-solving model grammar in Table 4.2.

Table 5.4: Sample *Grammar* file.

```
(defun dataset-portrial-grammar (portrial)
  (declare (ignore portrial))
  (create-grammar
   (list (create-rule 'compute-result '(b ac bd a) 'type-result .60)
         (create-rule 'compute-result '(b bd ac a) 'type-result .40)
         (create-rule 'type-result    '(result)    nil          .75)
         (create-rule 'type-result    '()          nil          .25))))
```

The four files are loaded into the system when the experiment is opened. Only the *Main* and *Stimulus* files are necessary to run EyeTracer and view protocols. The *Areas* file is needed for exploratory data analysis in which fixations are mapped to intended targets. The *Grammar* file is necessary for confirmatory analysis of the protocols with respect to a process model.

Using EyeTracer

Once the user has set up EyeTracer protocol files and experiment configuration files, EyeTracer is ready to perform eye-movement data analysis. On startup, EyeTracer creates a menu that allows the user to open an experiment data set. After a data set is opened, EyeTracer displays the menu shown in Figure 5.1. The first group of menu items open dialog boxes for viewing protocols, post-processing protocols, and setting parameter values. The upcoming sections describe these functions in detail. The second group of items allows the user to open or close a data set; the menu item "Close Equation" indicates that the "Equation" data set is currently open. The third group of items open submenus for selecting Lisp files pertaining to the data set: "EyeTracer Code" includes the *Main*, *Stimulus*, *Areas*, and *Grammar* files described earlier; and "Analysis Code" includes arbitrary files for manipulating and examining post-processed protocols.

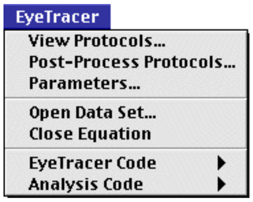
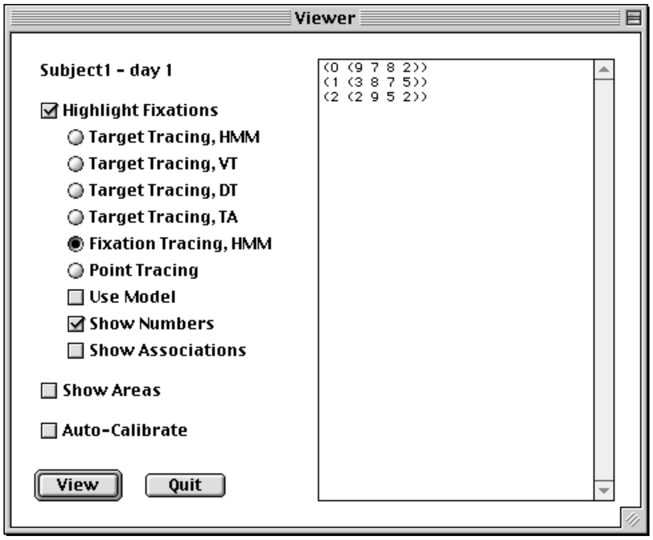


Figure 5.1: EyeTracer menu.

Viewing Protocols

Regardless of the type of data analysis, visualization of protocols is essential to assist in understanding behavior and developing cognitive models. EyeTracer provides a number of ways to visualize, or view, eye-movement protocols. When a user selects a set of protocols to view (i.e., a particular subject and day), EyeTracer opens a Viewer window as shown in Figure 5.2. On the left-hand side, the Viewer window displays a list of options for viewing protocols in various ways. On the right-hand side, the Viewer window displays the protocols that can be viewed; each protocol is represented by the number (0-2) and stimulus (a, b, c, and d of the equation) fields of the protocol `portrial`. To view a protocol, the user selects the protocol along with appropriate options and clicks on the "View" button.



Subject1 - day 1

☑ Highlight Fixations
  ○ Target Tracing, HMM
  ○ Target Tracing, VT
  ○ Target Tracing, DT
  ○ Target Tracing, TA
  ● Fixation Tracing, HMM
  ○ Point Tracing
  ☐ Use Model
  ☑ Show Numbers
  ☐ Show Associations

☐ Show Areas

☐ Auto-Calibrate

```
(0 (9 7 8 2))
(1 (3 8 7 5))
(2 (2 9 5 2))
```

Figure 5.2: Viewer window.

Sample protocols for the various option combinations are shown in Figure 5.3. If "Highlight Fixations" is off, EyeTracer defaults to (a) a velocity-based highlighting where points with smaller velocities are drawn larger to emphasize low-velocity fixations.[4] If "Highlight Fixations" is on, EyeTracer uses the selected tracing method to find fixations and emphasize fixation points. The tracing methods include six combinations of fixation identification and tracing algorithms: target tracing with the four identification algorithms I-HMM, I-VT, I-DT, and I-TA; fixation tracing with I-HMM; and point tracing. When "Use Model" is off, EyeTracer performs fixation identification only and does not trace the protocol. The system can display such protocols in several ways depending on the "Show Numbers" and "Show Associations" options; these ways include views (b) with only fixation highlighting, (c) with highlighting and numbering, and (d) with highlighting, numbering, and the associated targets (i.e., the nearest targets). When "Use Model" is on, EyeTracer traces the protocol with the given process model

---

[4] The algorithm for velocity-based highlighting came from the EPAL eye-movement data collection and analysis software developed by Scott Douglass.

and (e) displays predicted fixations in blue with their associated targets and unpredicted fixations in red with no associated target; the system also outputs the protocol trace to the Listener window for inspection. The “Show Areas” option displays the target area rectangles on-screen. The “Auto-Calibrate” option re-calibrates the protocol; this option is discussed as a future extension in the final chapter.

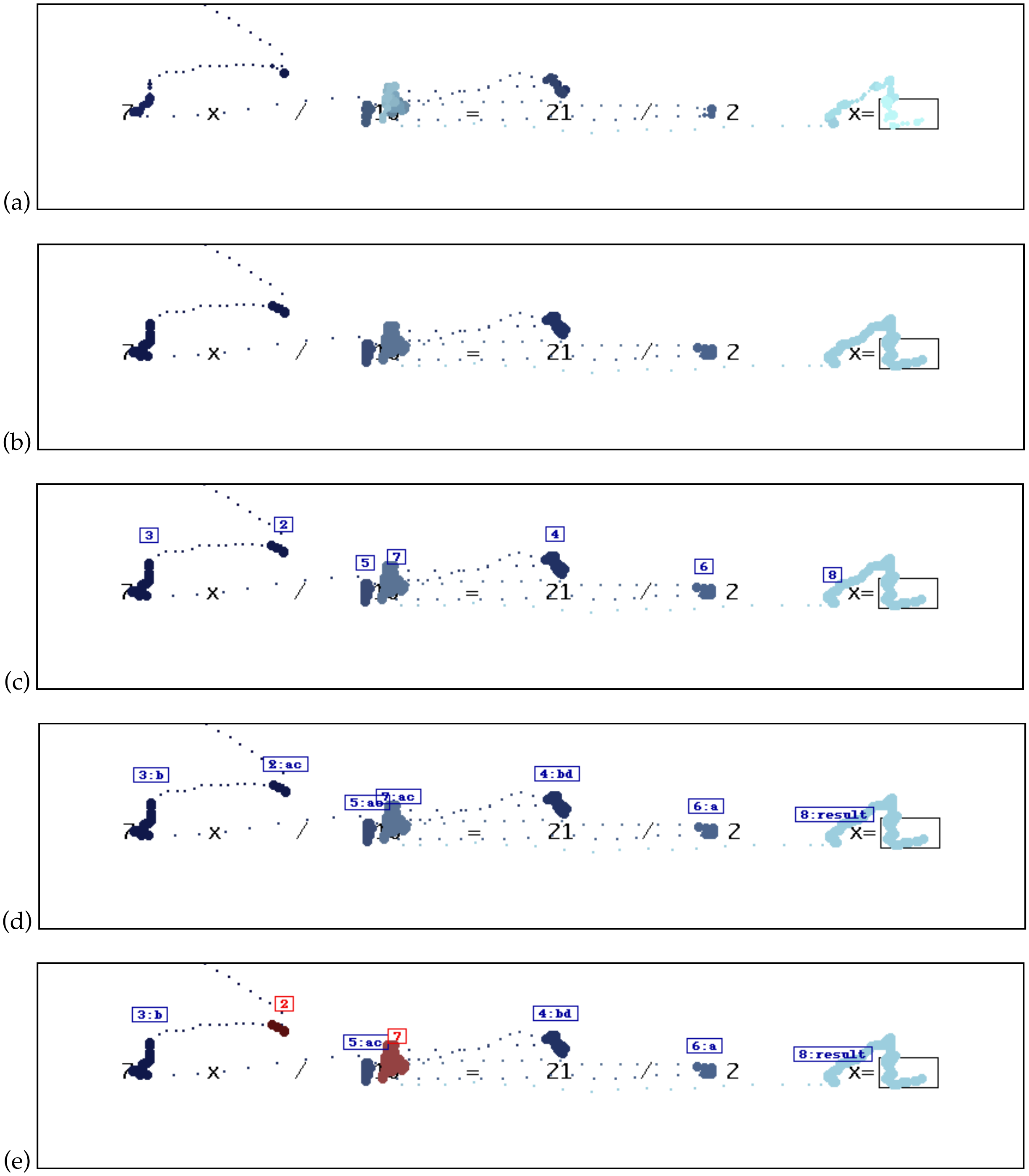
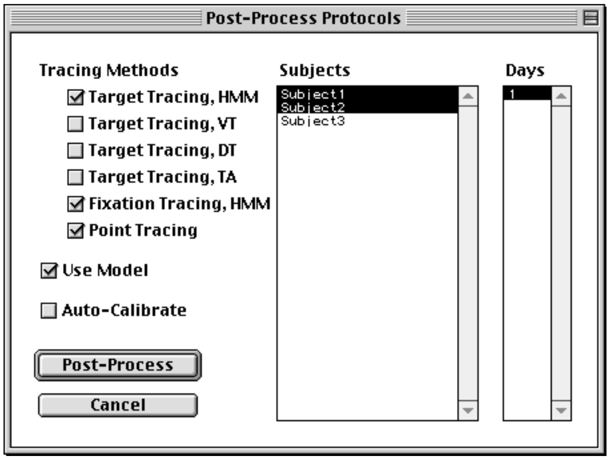


Figure 5.3: Visualizations of a sample protocol with (a) velocity-based highlighting, (b) fixation highlighting, (c) numbered fixations, (d) numbered fixations with target associations, and (e) traced fixations with blue shading and associations for predicted fixations and red shading for unpredicted fixations.

The various ways of viewing protocols makes EyeTracer an excellent tool for exploratory and confirmatory analysis with several levels of post-processing. Velocity-based highlighting adds only minimal processing to the protocol and thus presents an unbiased picture of subject behavior. Fixation highlighting adds slightly more processing (i.e., fixation identification) but improves the clarity of the protocol and further emphasizes fixations. Fixation numbering and associations also facilitate viewing by providing sequential information and naive target assignments, respectively, to the fixations. Tracing provides a view biased to the specific process model used, which assists greatly in finding mismatches between observed behavior and model predictions. Chapters 6-8 utilize all these views in creating, developing, refining, and comparing cognitive process models for the various application domains.

Post-Processing Protocols

While viewing protocols provides a glimpse of subject behavior at a detailed level, post-processing allows analysis of behavior across a large number of protocols. For post-processing, EyeTracer presents the Post-Process window shown in Figure 5.4. In the Post-Process window, the user selects any number of the various tracing methods with which to trace the protocols. The user also selects whether or not to use the model (i.e., trace or only identify fixations) and auto-calibrate (discussed in the final chapter). Finally, the user selects the subjects and days to be processed and clicks the "Post-Process" button.



☑ Target Tracing, HMM
☐ Target Tracing, VT
☐ Target Tracing, DT
☐ Target Tracing, TA
☑ Fixation Tracing, HMM
☑ Point Tracing

☑ Use Model

☐ Auto-Calibrate

Subject1
Subject2
Subject3

1

Figure 5.4: Post-Process window.

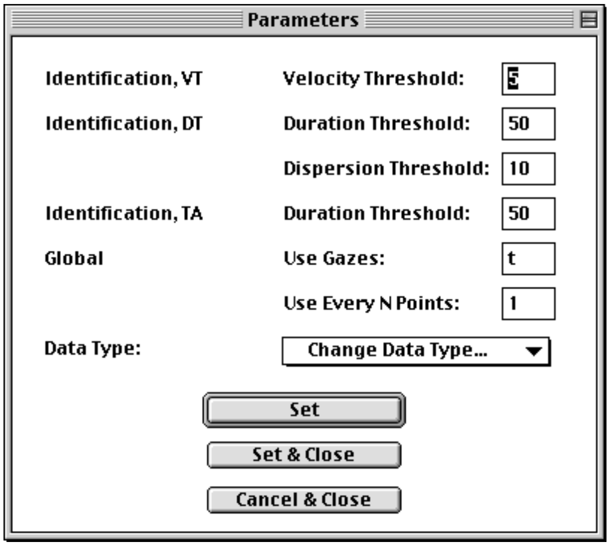
Post-processing generates a post-processed file for each protocol file. The post-processed file uses data structures `ppfile` and `pptrial` analogous to the `porfile` and `portrial` structures described earlier. The `ppfile` structure is identical to the `porfile` structure except that the `portrials` field is replaced with a `pptrials` field. The `pptrial` structure has a `number`, `stimulus`, and `response` fields identical to those in the `portrial` structure. In addition, it contains the following fields:

- `fixations`: list of < *target* , *duration* > fixation tuples
- `strategy`: list of targets as dictated by the model grammar rules
- `evaluation`: goodness-of-fit value

The resulting `ppfiles` can then be analyzed in any number of ways, such as finding average fixation durations, strategy choice probabilities, or average goodness-of-fit across protocols.

Setting Parameters

When analyzing protocols for viewing or post-processing, a user may wish to change parameters of the various tracing methods and general aspects of the data. The Parameters window shown in Figure 5.5 allows for such changes. The first group of four parameters determine the respective parameters for the identification algorithms I-VT, I-DT, and I-TA. The "Use Gazes" parameter indicates whether to use gazes or fixations as the standard unit of identification and tracing (see the discussion in Chapter 4). The "Use Every N Points" parameter allows for use of only a fraction of the actual eye-movement data to increase tracing efficiency; a value of *n* indicates that only every *n*th point will be used in tracing, discarding all other points. The "Data Type" parameter determines the data type of the experiment protocols, as defined by the eye-tracking equipment, sampling rate, and possible time averaging.



Identification, VT	Velocity Threshold:	5
Identification, DT	Duration Threshold:	50
	Dispersion Threshold:	10
Identification, TA	Duration Threshold:	50
Global	Use Gazes:	t
	Use Every N Points:	1
Data Type:	Change Data Type...	

Figure 5.5: Parameters window.

Chapter 6

Equation Solving

Introduction

The previous three chapters have described a class of algorithms for tracing eye-movement protocols and a working system, EyeTracer, that embodies these algorithms. This and the next two chapters employ and evaluate the algorithms in three application domains: equation solving, reading, and eye typing. Each domain provides an interesting test for the algorithms with respect to different aspects of tracing. As a whole, the domains represent a rigorous test set designed to illustrate the effectiveness of the tracing algorithms and their applicability to a wide range of domains.

This chapter applies the tracing algorithms to the domain of equation solving. While a number of researchers have studied mathematical reasoning in various contexts (e.g., Hegarty, Mayer, & Green, 1992; Kintsch & Greeno, 1985; Novick & Holyoak, 1991; Ohlsson, Ernst, & Rees, 1992; Zhu & Simon, 1987), only a few have incorporated the analysis of eye movements in their studies (e.g., DeCorte, Verschaffel, & Pauwels, 1990; Suppes, 1990; Suppes et al., 1983; Verschaffel, DeCorte, & Pauwels, 1992). These latter studies have shown that eye movements can reveal a great deal about human reasoning in mathematical settings. This chapter discusses how the automated tracing methods proposed in the thesis can assist in eye-movement data analysis and reveal interesting behavior hidden in previously-used types of analysis.

The equation-solving task, as discussed in earlier chapters, entails solving an equation of the form $b\,\mathbf{x} \,/\, ac = bd \,/\, a$ and computing the value of $\mathbf{x}$ as $cd = (ac/a)(bd/b)$. Anderson and Douglass (prep) collected eye movements for college students performing this task as a way to understand the interaction of visual attention and problem solving. They were especially interested in what strategies subjects employed to encode the components of the equation and how they interleaved encoding with computation of results (e.g., the values of *c* and *d*). Although the eye movements provided some aid in understanding, the variability and noise in the protocols, combined with the large amount of data, made analysis extremely difficult and provided ambiguous results. It was clear that more sophisticated automated analysis techniques were needed to alleviate variability and noise and to examine large numbers of protocols quickly and accurately.

This chapter describes two studies of the equation-solving task. The first study evaluates the proposed tracing methods and compares their performance to that of human coders. The study examines task data collected in a "constrained" problem-solving setting: subjects were taught a particular strategy for solving the equations and asked to solve the equations using only this strategy. The constraint allows us to know, at least to a reasonable degree, what cognitive processes were involved in the observed subject behavior. With these data, the study compares the performance of the tracing methods to that of expert human coders to better gauge the difficulty of interpretation. In addition, the study examines how quickly and accurately the various tracing methods interpret subject protocols and compares the methods with respect to one another.

The second study investigates how to develop and refine a cognitive model of subject behavior in an "unconstrained" equation-solving task. The unconstrained data, which comes from Anderson and Douglass' original study, describe subjects' behavior when they solved equations in whatever way they desired. The study demonstrates how the tracing methods facilitate exploratory and confirmatory analysis of the data and development of an ACT-R (Anderson & Lebiere, 1998) cognitive process model. The study begins with an exploratory analysis of subject protocols to understand their behavior at a fine-grain level. The exploratory analysis helps to generate an initial prototype model grammar for the task. The study continues with several iterations of confirmatory analysis in which the model grammar is tested against observed data and refined accordingly. Finally, the model grammar is translated to an ACT-R model that makes detailed predictions about the cognitive steps, eye movements, and keystrokes embodied by the model grammar.

Study 1: Constrained Equation Solving

Study 1 provides a rigorous test of the tracing methods in the constrained equation-solving task. The tracing methods determine the best interpretation for a given protocol with a given process model that describes some set of problem-solving strategies. Because we can never be sure of the underlying cognitive processes that generated a sequence of eye movements, we cannot describe the "accuracy" of the tracing methods with absolute certainty. However, we can provide strong clues to their accuracy in two ways. The first way involves comparing their interpretations to the "correct" interpretations as generated in an *instructed-strategy paradigm*. The instructed-strategy paradigm asks subjects to solve problems using a particular strategy. Before starting the task, the experimenter instructs the subjects as to the exact sequence of steps that they should perform in solving the problem; the steps include visual accesses (i.e., look at and encode this value), computational cognitive processes (e.g., divide ac by a to get c), and motor actions (e.g., type the result). The instruction provides a "correct" answer for interpreting the protocol—that is, we know what strategy a subject was employing for a given protocol. By asking each tracing method to interpret the protocols as one of the instructed strategies, we can compare their interpretations to the "correct" strategies and determine how accurately they perform.

The second way of testing the tracing methods entails comparing their performance to that of expert human coders. Regardless of the power of the tracing methods, it is reasonable to assume that they may not interpret all protocols accurately because of subject variability and/or data noise. However, since the tracing methods are intended to automate the analysis task typically performed by humans, we are primarily interested in how accurately they interpret compared to human coders. This comparison controls for protocols in the data set that simply cannot be interpreted because of the subject failing to perform the instructed strategy or because of extreme data noise.

The constrained equation-solving study uses the instructed-strategy paradigm to compare the tracing methods to one another and to human coders. In the task, subjects solved equations in five one-hour sessions. In the first session, subjects could solve the equations however they wished. In each of the next four sections, subjects were instructed to solve the equations using one of the strategies in Table 6.1, which can be generally described as follows: (1) read values left-to-right, (2) read values left-to-right in pairs, (3) read values right-to-left, and (4) read values right-to-left in pairs. In all strategies, intermediate results (c and d) are computed as soon as possible. These strategies were chosen to provide an initial window into one of the original goals of the equation-solving experiment, namely whether eye movements provide information as to when computation occurs: The encodings of ac and bd appear in various combinations of ordering and computation, such that computation occurs during one encoding or the other, as shown in boldface in the table. In addition, each strategy is very similar to one other strategy (1 and 2, 3 and 4), making interpretation sometimes difficult for the tracing methods and human coders.

Table 6.1: Instructed equation-solving strategies for the constrained task. Encodings during which computation occurs appear in boldface.

Number	Strategy
1	encode *b*, encode *ac*, **encode *bd*,** compute *d*, **encode *a*,** compute *c*, compute **x**, type **x**
2	encode *b*, encode *bd*, compute *d*, encode *ac*, **encode *a*,** compute *c*, compute **x**, type **x**
3	encode *a*, encode *bd*, **encode *ac*,** compute *c*, **encode *b*,** compute *d*, compute **x**, type **x**
4	encode *a*, **encode *ac*,** compute *c*, encode *bd*, **encode *b*,** compute *d*, compute **x**, type **x**

Method

Subjects

Five Carnegie Mellon students, all men, with normal, uncorrected vision participated in the five-day experiment. Because of extreme data noise during eye tracking, one subject was omitted from data analysis. The remaining four subjects were assigned in days 2-5 to the strategies in Table 6.1 as follows: subject S1, strategies 2-3-1-4; S2, 3-1-4-2; S3, 1-4-2-3; S4, 2-3-1-4.

Materials

The constrained equation-solving experiment was designed to be as similar to the original experiment (described in Study 2) as possible. Subjects solved problems in four conditions: eye-tracking, mouse-tracking, eye-blocked-tracking, and control. In the eye-tracking condition, subjects solved equations as their eye movements were tracked. In the mouse-tracking condition, subjects moved the mouse over components of the equation to uncover opaque blocks over those components; this allowed for study of an alternate, less noisy method of tracking visual attention. In the eye-blocked-tracking condition, subjects used their eye movements to uncover the same opaque blocks; the pointer was required to be over a particular block for 100 ms before it was removed to alleviate incidental uncoverings. In the control condition, no eye movements were tracked. In addition, the problems were equally divided between *complex* problems of the form $b\,\mathbf{x} \,/\, ac = bd \,/\, a$ and *simple* problems of the form $\mathbf{x} \,/\, c = d$. Values for a, b, c, and d used to create the stimuli were generated randomly within the range [2,9]. Because this study focuses on evaluating the eye-movement tracing methods, the following analyses focus only on the eye-tracking condition. The mouse-tracking condition will be discussed in the unconstrained equation-solving study. The control condition revealed no significant variation from the eye-tracking condition; in other words, the eye tracking itself did not affect subject behavior. The eye-blocked-tracking condition was generally difficult for subjects and thus generated very sporadic, unreliable data. The simple problems resulted in uninteresting eye movements (often single fixations) and are not discussed.

Eye-movement data were collected using an Iscan RK726/RK520 eye tracker with a Polhemus FASTRACK head tracker. The Iscan eye tracker uses the pupil-center and corneal-reflection points to estimate the angle of the left eye's gaze. The head tracker determines the position of the head and eye with respect to the visual field. The system collects eye and head information to produce a stream of on-screen point-of-regard locations. Data were sampled at a rate of 120 Hz. Subjects were seated approximately 30" from the computer display screen. Characters in the stimuli were approximately 1/4" in height (0.48° of visual angle) with approximately 1" (1.91° of visual angle) between them.

Procedure

The experiment comprised five one-hour sessions on consecutive days. On each day, the experimenter calibrated the eye-tracking equipment to the particular subject and checked the quality of the data. On the first day, the experiment introduced the subjects to the equipment and explained the procedure. The subject then solved 192 problems (48 in each of 4 conditions, half complex and half simple) in whatever way they desired. On days 2-5, the experimenter taught the subject one of the four Table 6.1 strategies for solving the equations. This included an explanation of the order in which the values were to be encoded and the time at which intermediate and final values were to be computed. The subject proceeded to solve 192 problems using that strategy. Note that the subject employed only a single strategy per day to avoid confusion between strategies.

Each trial began with the subject fixating a box in the upper-left corner of the screen; this fixation ensured that subjects' eyes fell in approximately the same location at the start of the trial. The subject proceeded to solve the problem and type the result on the numeric keypad (on a standard extended keyboard). The subject hit the enter key to end the trial.

Tracing Setup

The following analyses for the constrained equation-solving study utilize the six combinations of identification and tracing methods implemented in the EyeTracer system described in Chapter 5: target tracing with I-HMM, I-VT, I-DT, and I-TA; fixation tracing with I-HMM; and point tracing. We refer to these combinations as Target-HMM, Target-VT, Target-DT, Target-TA, Fixation-HMM, and Point, respectively. The analyses center on the comparison of the interpretations as generated by the tracing methods and the "correct" interpretations as given by the instructed strategies. The analyses includes a total of 369 protocols: 24 clear-track protocols for complex problems for each of 4 subjects on each of 4 days, less 15 protocols that were deemed too noisy for reasonable analysis. All analyses aggregate fixations into gazes as discussed in Chapter 4.

The analyses require a specification of the target areas and the process model grammar. The target areas used in this chapter are shown in Figure 6.1. The target areas for *b*, *ac*, *bd*, *a*, and *result* represent gazes on the respective values. The middle areas (*ac*, *bd*, *a*) are approximately 2° of visual angle in width and 4° in height; the larger height was used to capture fixations above and below the values. The left-most and right-most areas (*b* and *result*) extend farther outward to compensate for acceleration toward the boundaries produced by the eye tracker. In addition, the target area *start* (the bottom of which can be seen above *b*) captures initial fixations on the start region that trigger each trial.

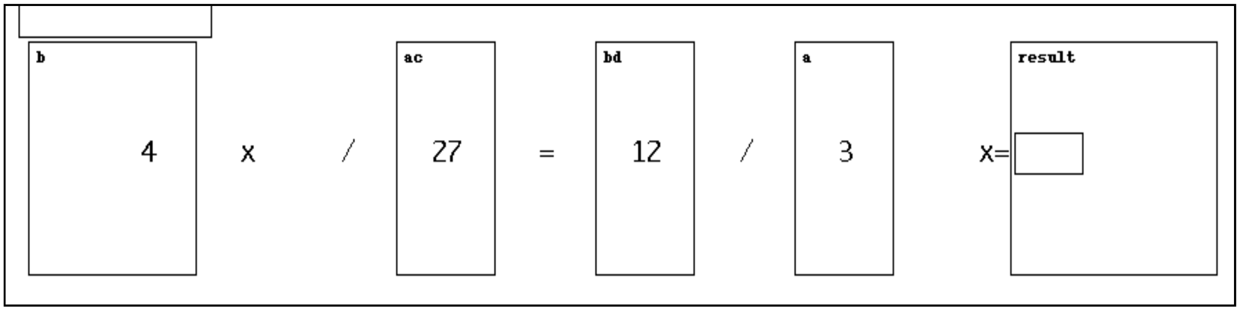


Figure 6.1: Equation-solving target areas.

To determine the process model grammar, the constrained experiment used the four instructed strategies shown in Table 6.1. These strategies translate in a straightforward manner to the grammar shown in Table 6.2. The grammar incorporates two additional features not included in the original strategies. First, rule 1 (the rule that fires first) produces a fixation on the *start* area, as subjects were required to do. Second, rules 6-8 fixate any item(s) as review (rule 6), fixate the *result* area and terminate (rule 7), or simply terminate (rule 8). Because subjects were instructed to perform the strategies with equal frequency, the rule probabilities are evenly distributed among rules for each subgoal.

Table 6.2: Model grammar for the constrained equation-solving study.

Number	Rule			Probability
1	**start-trial**	→	*start* **compute-result**	1
2	**compute-result**	→	*b ac bd a* **type-result**	1/4
3	**compute-result**	→	*b bd ac a* **type-result**	1/4
4	**compute-result**	→	*a bd ac b* **type-result**	1/4
5	**compute-result**	→	*a ac bd b* **type-result**	1/4
6	**type-result**	→	*any* **type-result**	1/3
7	**type-result**	→	*result*	1/3
8	**type-result**	→		1/3

Parameter Setting and Sensitivity Analysis

This first analysis of the equation-solving data determines good values for the various tracing and identification parameters and illustrates how sensitive tracing is to these parameter settings. Target-HMM has 8 parameters, namely the parameters of the two-state HMM used to decode velocities: 2 transition probabilities, 1 velocity mean, and 1 velocity variance for each of the two states. These parameters were estimated using the Baum-Welch reestimation procedure detailed in Appendix A. First, the transition parameters were initialized with random values and the mean and velocity parameters were set to reasonable values. Second, the parameters were iteratively trained over 40 sample protocols (approximately 19,000 data points) until the log probability for successive iterations differed by less than 1.0. In nine iterations, training converged to the two-state HMM shown in Figure 6.2. The velocity mean and variance for the saccade state were set to 13.36 and 72.89; those for the fixation state were set to 1.58 and 1.59, respectively. (Note that these distributions represent point-to-point velocities as measured in pixels for the given eye-tracker setup.) Prior probabilities for the two states were set to 0.5. Given this two-state HMM, Target-HMM interpreted 91.9% of protocols correctly (with respect to the instructed strategy). The average mismatch for these interpretations (i.e., sequence distance between the observed and predicted sequences) was .87. It is important to note that the 8 parameters required to achieve this accuracy were estimated automatically.

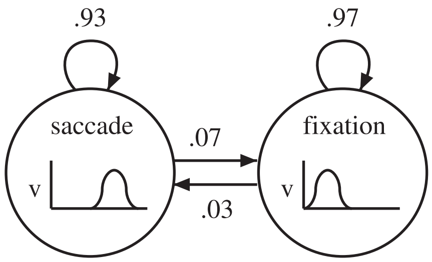


Figure 6.2: Two-state HMM used for I-HMM identification.

Target-VT requires only one parameter value, namely the point-to-point velocity value used as a threshold. From HMM reestimation for Target-HMM, we learned that velocities for fixations hover around 1 or 2 and those for saccades hover over 10. Thus, six different velocity thresholds were tested: 1, 2, 3, 4, 5, and 10. Table 6.3 shows the resulting performance for Target-VT with these values; the table entries show the percentage of protocols interpreted correctly along with the average mismatch in parentheses. Clearly Target-VT is not particularly sensitive to the setting of the velocity threshold, maintaining accuracy between 87% and 89% across parameter values. While other values performed equally well, brief qualitative analysis revealed that the lowest thresholds produced what seemed like too many fixation points, and the highest thresholds, too many saccade points. The value 5 was chosen as the default value for the velocity threshold primarily because it falls approximately halfway between the velocity means for fixations and saccades.

Table 6.3: Percent correct and mismatch (in parentheses) for Target-VT over various velocity thresholds (in pixels).

Velocity	1	2	3	4	5	10
Result	88.1 (.95)	88.3 (1.06)	87.5 (1.23)	87.5 (1.16)	87.8 (1.63)	88.9 (1.09)

Target-DT has two parameters: the dispersion threshold that determines the maximum dispersion (distance between farthest points) for a group of fixation points, and the duration threshold that determines the minimum fixation duration. Three dispersion-threshold and four duration-threshold values were tested across the protocol data set. Table 6.4 shows the resulting accuracy (percent correct) and average mismatch for all combinations. Target-DT behaved in a fairly robust way for different dispersion thresholds. However, the duration threshold had significant effects, with longer durations generally performing worse. Part of Target-DT's parameter sensitivity comes from the interaction between the parameters: When the dispersion is low but the duration high, it is difficult to find a group of fixation points that fit within the dispersion but still last the duration. The values of 10 and 50 were chosen as the default values for the dispersion and duration thresholds, respectively, because of the excellent performance at and near that combination.

Table 6.4: Percent correct and mismatch (in parentheses) for Target-DT over various dispersion thresholds (in pixels) and duration thresholds (in ms).

Dispersion	Duration			
	50	100	150	200
10	91.3 (.82)	81.0 (1.16)	53.4 (2.18)	33.1 (3.28)
20	90.0 (.92)	89.4 (.84)	79.4 (1.26)	64.0 (1.83)
30	87.5 (.93)	90.8 (.79)	85.6 (.99)	76.4 (1.44)

Target-TA requires only a duration threshold for the minimum duration of a fixation over a target area. The sensitivity test used the same duration-threshold values as that for Target-DT. The setting of the duration threshold drastically affects performance for Target-TA, as shown in Table 6.5. The value of 50 was chosen as the default duration threshold because of good performance for this value and parsimony with the analogous parameter for Target-DT.

Table 6.5: Percent correct (mismatch) for Target-TA over various duration thresholds.

Duration	50	100	150	200
Result	87.8 (.97)	81.8 (1.17)	73.7 (1.38)	69.4 (1.65)

Fixation-HMM requires the specification of a two-state HMM for fixation identification and fixation HMMs for tracing. Parameters for the two-state HMM can be taken from the Target-HMM analysis above. To specify the fixation HMMs, various forms of HMMs resembling those in Chapter 4 were tested on the data set. The accuracy of the HMMs in these tests ranged approximately from 92.4% to 93.2%. The fixation HMM shown in Figure 6.3 performed best (93.2%) and was chosen as the default fixation HMM for this task. This HMM allows repeated fixations on the target area to model gazes with multiple fixations. It also allows for incidental fixations occurring before or after the target fixation with low probability.

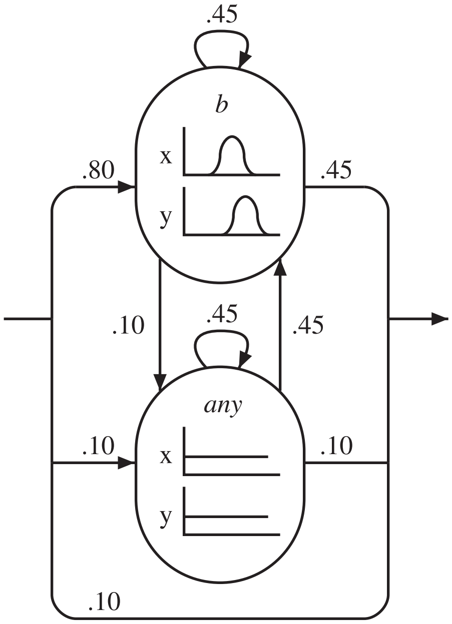


Figure 6.3: Equation-solving fixation HMM for target area *b*.

Point, like Fixation-HMM, requires the specification of fixation HMMs for tracing. Again, various possible fixation HMMs were evaluated based on their accuracy on the protocol set. The HMMs produced performance between 93.0% and 94.3%. The fixation HMM that performed best (94.3%), shown in Figure 6.4, was chosen as the default fixation HMM. The transition probabilities and v means and variances are identical to those for the two-state velocity HMM in Figure 6.2. The x and y means and variances for the fixation (second) state are derived from the target area: the means center around the area x and y centers, and the variances are computed as the square of half the width or height (i.e., the size of the target area represents two standard deviations, one in either direction off the mean). This HMM allows skipping of the fixation, but does not allow incidental fixations; apparently the ability to decode incidental fixations is compensated by the saccade state, since incidental fixations can be interpreted as saccades.

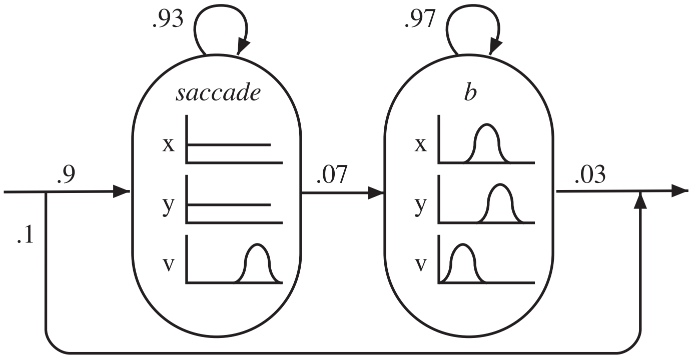


Figure 6.4: Equation-solving fixation HMM for target area *b*.

In addition to the study of the method-specific parameters discussed above, it is interesting to note the sensitivity of the methods to setting of the global parameter that traces only every *n*th point (see Chapter 5). Table 6.6 shows the accuracy and mismatch for all six method combinations when tracing using every *n*th point for values of n = 1, 3, 5, and 10.[5] While performance clearly degrades with fewer data points, some of the algorithms—notably Target-HMM, Target-VT, Fixation-HMM, and Point—are remarkably robust with respect to this parameter. In addition, the removal of points in this manner generally lead to a speedup in tracing of approximately a factor *n*. Potential speedups such as this one are discussed more fully in the final chapter.

Table 6.6: Percent correct (mismatch) for all method combinations using every *n*th point.

n	T-HMM	T-VT	T-DT	T-TA	F-HMM	Point
1	91.9 (.87)	87.8 (1.16)	91.3 (.82)	87.8 (.97)	93.2 (12.0)	94.3 (15.3)
3	92.1 (.78)	90.8 (.90)	63.4 (1.48)	80.8 (1.12)	87.8 (12.0)	94.0 (16.4)
5	89.7 (.82)	92.4 (.79)	61.0 (1.63)	84.3 (1.05)	87.3 (12.0)	93.8 (16.8)
10	85.9 (.99)	87.8 (.96)	23.8 (4.00)	68.8 (1.38)	84.3 (12.1)	86.7 (17.3)

Evaluations and Comparisons

Given the parameter settings described above, we now evaluate and compare the performance of the tracing methods among themselves and with human coders. We first

---

[5] Generating the results in Table 6.6 for n={3,5,10} required reestimation of certain parameters: HMM velocity distributions and transition probabilities for Target-HMM, Fixation-HMM, and Point were retrained from a training set of modified protocols; the velocity threshold for Target-VT was changed to {10,15,20}; and the dispersion threshold for Target-DT was changed to {30,50,50}.

examine performance of the methods and coders for a test set of the collected protocols. The test set of protocols comprised the final two protocols from each session for each of the four subjects, 32 protocols in all. This test set was given to two human coders as printed eye-movement displays with numbered fixations (see Chapter 5). The coders classified each protocol as one of the four strategies in Table 6.1. Both coders (a professor and graduate student in cognitive psychology) were highly experienced in studying such printed displays and in developing cognitive process models. The same protocols were given to the six tracing methods, which also classified each protocol as one of the four strategies.

Table 6.7 shows the percent agreement between each tracing method and human coder, along with their correctness as determined by whether their interpretations match the instructed strategies. The target-based tracing methods produce moderate accuracy, with Target-VT (81.2%) having somewhat less accuracy than the others (87.5%). The HMM-based tracing methods perform extremely well, achieving an accuracy of 93.7%. Of the human coders labeled HC1 and HC2, HC2 gives high accuracy (90.6%) while HC1 gives the least accurate interpretations of all the methods (78.1%). With respect to between-method agreement, the target-based methods agree reasonably well with one another (84.4-96.9%) but less well with the HMM-based methods (81.2-84.4%). The HMM-based methods show high agreement (93.7%). Interestingly, the human coders exhibit low agreement (81.2%); in fact, with only one exception, the human coders agree as well or better with the automated methods than with each other. Overall, this analysis clearly shows the automated tracing methods perform with high accuracy and their performance is almost indistinguishable from that of expert human coders.

Table 6.7: Percent agreement between each tracing method, human coder, and "correct" answer as indicated by the instructed strategy.

Method	Automated						Human		Correct
	T-HMM	T-VT	T-DT	T-TA	F-HMM	Point	HC1	HC2	
T-HMM	–	87.5	87.5	96.9	84.4	84.4	81.2	90.6	87.5
T-VT		–	84.4	90.6	84.4	84.4	81.2	84.4	81.2
T-DT			–	90.6	81.2	84.4	84.4	90.6	87.5
T-TA				–	84.4	84.4	84.4	93.7	87.5
F-HMM					–	93.7	81.2	84.4	93.7
Point						–	78.1	90.6	93.7
HC1							–	81.2	78.1
HC2								–	90.6

To better illustrate how the automated methods and the human coders interpreted the test set protocols, we now analyze several protocols in detail to provide a more detailed picture of their operation. Figure 6.5 shows a protocol for the instructed strategy [*b ac bd a*] (as named by the order of value encodings) that all methods and coders classified correctly. The protocol contains an initial incidental fixation (2) that presumably represents a missed fixation on target *b*. However, the remainder of the protocol clearly represents a left-to-right strategy, and all methods and coders recognize it as such. As an aside, this protocol nicely illustrates another property of many of these protocols: the increased durations of fixations during which computation takes place. Fixations 5 and 6, during which *c*, *d*, and possibly *result* are computed, are notably larger (and longer) than fixations 2-4. The next section specifically analyzes the effects of computation on gaze duration.

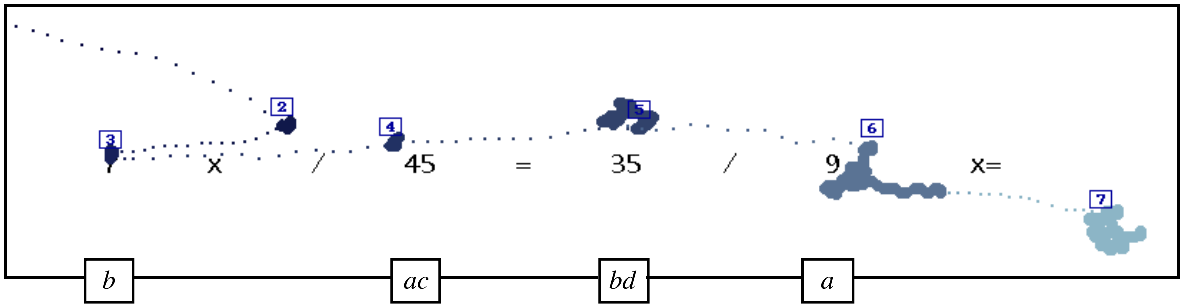


Figure 6.5: Protocol for [*b ac bd a*] correctly classified by all methods and coders.

Figure 6.6 shows a protocol for the strategy [*b bd ac a*] that all methods and coders except HC1 interpreted correctly. However, many of the automated interpretations turned out to represent lucky guesses on the part of the methods. Besides the initial fixation on the *start* area, the protocol seems to include only two fixations, namely on *bd* and *ac*. However, there is clearly a bend near *b* where a fixation presumably occurred, but only Target-TA and Point recognized this fixation. (The lack of a clear fixation at this point is due primarily to time-averaged sampling.) The remaining four automated methods—Target-HMM, Target-VT, Target-DT, and Fixation-HMM—traced the fixation sequence [*start bd ac*] and guessed correctly that the protocol represents [*b bd ac a*], although [*a bd ac b*] matches this sequence equally well.

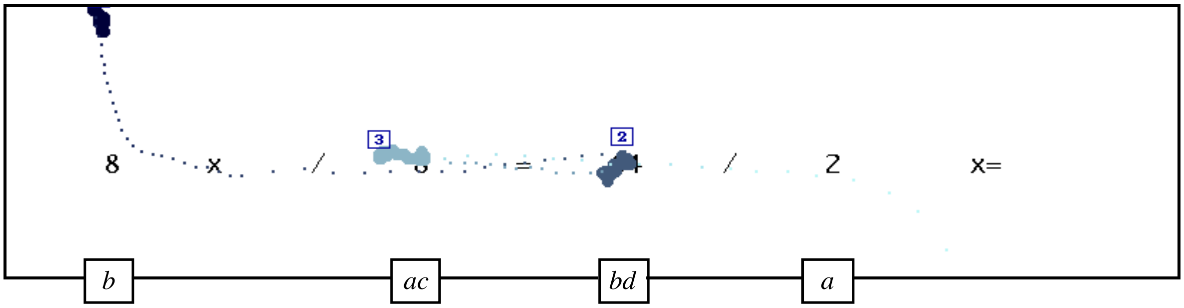


Figure 6.6: Protocol for [*b bd ac a*] correctly classified by all methods and coders except HC1.

Figure 6.7 includes a protocol for the strategy [*a ac bd b*] that only Fixation-HMM and Point classified accurately. The protocol has four small fixations that occur far off-center from their intended targets, and the fixations on *ac* and *a* are out of order: fixation 2 encodes *ac* and fixation 3 encodes *a*. Fixation-HMM interpreted fixation 2 as incidental; Point skipped the fixation by interpreting its points as saccade points. The target-based methods interpreted the protocol as [*a bd ac b*]—perhaps a reasonable classification given the order inversion. This result highlights a problem with all the automated methods: they are unable to recognize order inversions in the fixation sequence. In this case, however, the human coders also seemed unable to recognize the inversion, interpreting the protocol incorrectly as [*a bd ac b*].

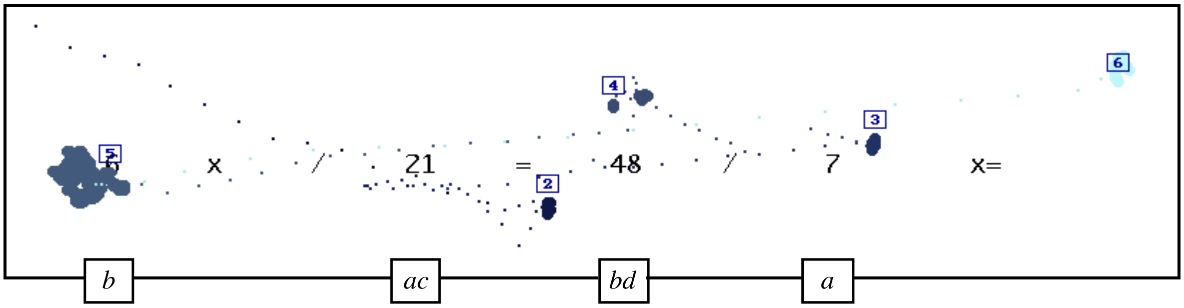


Figure 6.7: Protocol for [*a ac bd b*] correctly classified by Fixation-HMM and Point.

Figure 6.8 shows a protocol for the strategy [*a bd ac b*] that only Target-HMM, Target-VT, Target-DT, and HC2 classified accurately. The protocol is fairly confusing: fixation 2 presumably encodes *a*, fixation 3 encodes *bd*, and fixation 4 encodes both *ac* and *b* using parafoveal vision. In addition, there is an incidental fixation (5) that could represent a glance at the *result* area. The power of Fixation-HMM and Point actually hurt their interpretations in this case: they both interpreted the initial fixation in the *start* area (not shown) as a fixation on *b* and fixation 2 as incidental, in which case their classification of the protocol as the strategy [*b bd ac a*] seems very reasonable. The combined effect of the parafoveal encoding of *b* and the incidental fixation just over *a* caused these methods to interpret the protocol incorrectly.

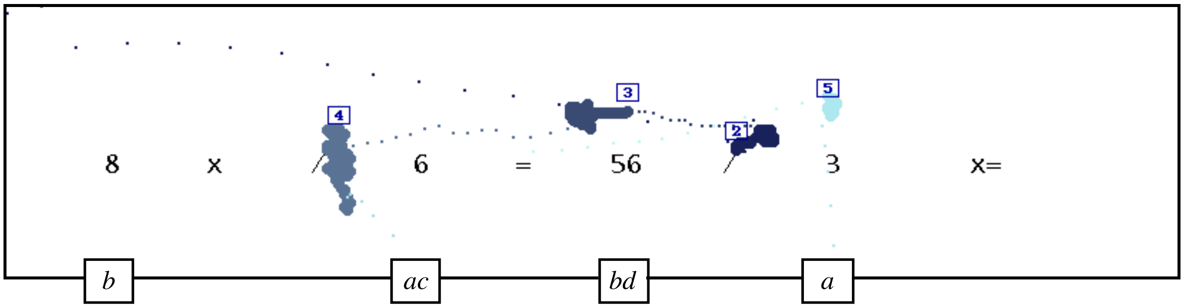


Figure 6.8: Protocol for [*a bd ac b*] correctly classified by Target-HMM, Target-VT, Target-DT, and HC2.

Figure 6.9 shows a protocol for the strategy [*a bd ac b*] that only Fixation-HMM classified correctly. This protocol illustrates, like some of the above protocols to a more limited extent, how some subject protocols simply do not correspond to the instructed strategy. The protocol begins with a fixation on *a* (2) followed by a back-and-forth motion over the final three values: *ac* (3), *bd* (4), *a* (5), *bd* (6), and *ac* (7). The protocol ends with a fixation on *result*. Clearly the protocol does not conform to the instructed strategy, which dictates a purely right-to-left scanning sequence. Although one method classified the protocol correctly, it is difficult to fault the other methods and human coders with a misclassification—the protocol exhibits too much variability from expected strategies to be interpreted in a reasonable way.

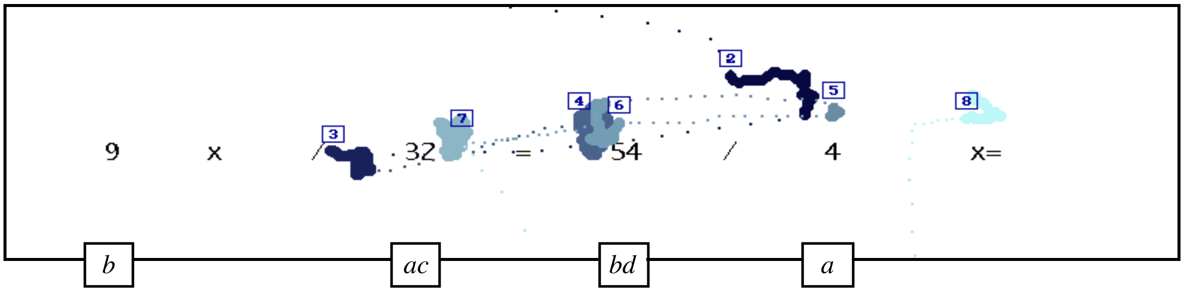


Figure 6.9: Protocol for [*a bd ac b*] correctly classified only by Fixation-HMM.

The above analysis of individual protocols provide a somewhat distorted picture of the performance of the methods and coders. While the protocols above illustrate a number of erroneous interpretations, it should be stressed that overall the methods and coders performed moderately to very well in classifying the protocols (78.1-93.7%). In addition, the high agreement between the interpretations and the instructed strategies suggests that the instructed-strategy paradigm used in this study is extremely effective and could facilitate development of data analysis methods in other contexts.

The analysis until now has used only the 32 protocols in the test data set. A much better comparison can be produced when considering all the protocols in the data set (369 protocols in all); because of the tedium of having human coders interpret so many protocols, the analysis compares only the automated methods. Table 6.8 shows the percent of correct protocols for all the automated methods using the default parameters described in the previous section. Point gave the best performance (94.3%), with Fixation-HMM close behind (93.2%). The target-based methods were slightly less accurate: Target-HMM and Target-VT produced fairly accurate interpretations (91.9% and 91.3%), while Target-VT and Target-TA gave the lowest performance (87.8%).

Table 6.8: Percent correct for the automated methods over all protocols.

Method	T-HMM	T-VT	T-DT	T-TA	F-HMM	Point
Accuracy	91.9	87.8	91.3	87.8	93.2	94.3

While the automated tracing methods and human coders interpreted the protocols with similar accuracy, the tracing methods generated interpretations significantly faster. Table 6.9 shows the average time in seconds needed to code a single protocol for each tracing method and human coder. The entries for the automated methods represent tracing times on a 200 MHz Power Macintosh; times for Fixation-HMM and Point do not include the (small) overhead cost of building the tracer HMM. The table shows that while the human coders required approximately one minute per protocol, the tracing methods performed at least an order of magnitude faster. Point was by far the slowest of the automated methods, whereas the others exhibited similar times with Target-VT and Target-TA being the fastest. In summary, automated tracing can produce speedups of 10 to 100 or more over conventional human coding. Of course, the time needed for tracing depends highly on the size of the protocol and the complexity of the process model used. However, this study suggests that automated tracing can generate interpretations up to and faster than real-time when speed is critical. In addition, the time results in Table 6.9 combined with the accuracy results in Table 6.8 illustrate a real speed-accuracy tradeoff in using the tracing methods: the most powerful methods produce more accurate but slower interpretations, and the least powerful methods produce less accurate but faster interpretations.

Table 6.9: Time needed to analyze a single protocol, in seconds.

Method	T-HMM	T-VT	T-DT	T-TA	F-HMM	Point	HC1	HC2
Accuracy	.038	.017	.030	.017	.045	1.01	67.5	60.0

Study 2: Unconstrained Equation Solving

Study 2 demonstrates how the tracing methods facilitate analysis and understanding of eye-movement behavior and assist in the development of cognitive process models. The study examines data collected by Anderson and Douglass (prep) concerning how people interleave visual attention and computation when solving equations in an "unconstrained" setting—that is, when they can solve the equations in whatever manner they desire. This study, like the original Anderson and Douglass (prep) study, focuses on two primary issues concerning behavior in this task: strategy use and computation. Strategy use describes what visual strategies subjects employ when encoding the various components of the equation. This issue emphasizes the sequential nature of subject behavior, namely the order of the gazes on the equation components. Computation describes how subjects decide to interleave encoding of the equation components and computation of the intermediate results (i.e., c and d). This issue emphasizes the temporal aspect of the protocols, in that we can identify computation by demonstrating an effect on gaze durations for gazes during which such computation might take place. While other aspects of subject behavior besides strategy use and computation would certainly be interesting to analyze (e.g., trial response times), this study focuses on these aspects to emphasize analyses that tracing can facilitate and improve.

The unconstrained equation-solving data represent subjects' visual attention using two experimental conditions: an eye-tracking condition, in which subjects' eyes were tracked; and a mouse-tracking condition, in which they uncovered opaque blocks over components of the equation using a mouse pointer. The two conditions allow us to analyze visual attention behavior with different modalities and compare the relative noise and variability. Previous work, most notably of Lohse and Johnson (1996), has found that mouse-based data has generally much less noise and variability than eye-based data, and thus is easier to analyze and understand. However, mouse-based data collection can sometimes alter strategy selection in the task, making eye-based data preferable for more realistic analysis of behavior.

Method

Subjects

Eleven Carnegie Mellon students, two women and nine men, with normal uncorrected vision participated in the four-day experiment. One subject was omitted from analysis because of extreme difficulty solving the problems.

Materials

The materials in this experiment were identical to those in Study 1. However, the following analyses will address both the eye-tracking and mouse-tracking data to emphasize the similarities and differences between the two modalities.

Procedure

The procedure in this experiment was identical to that of Study 1 with two exceptions: the experiment comprised four one-hour sessions; and subjects were given no instructions as to how to solve the equations, allowing them to solve the equations in any manner they chose.

Results without Tracing

Before illustrating how tracing assists in analysis of the task data, we perform some preliminary analysis using more traditional eye-movement data analysis techniques. This section addresses data from both the mouse-tracking and eye-tracking conditions with respect to strategy use and computation. The mouse-tracking condition includes a total of 960 protocols: 24 protocols for complex problems for each of 10 subjects over 4 days. The eye-tracking condition includes an equal number, minus 68 protocols with severe data noise, for a total of 892 protocols. All analyses aggregate fixations into gazes as discussed in Chapter 4: For the mouse-tracking condition, a "fixation" represents an uncovering of an opaque block over one of the equation values b, ac, bd, and a; for the eye-tracking condition, fixations were identified using the I-HMM algorithm and assigned to the nearest target.

Mouse-Tracking Condition

Strategy Use. We first examine the mouse-tracking data to determine common encoding strategies. To understand subjects' strategy use without tracing, we can use a *frequency tree* that displays the various gaze sequences in a tree-like structure. Figure 6.10 shows the frequency tree for the mouse-tracking condition. Each line, or branch, of the tree includes a gaze prefix (i.e. sequence of gazes up to that point), the number of protocols that the prefix represents, and the average duration of the gaze (in brackets). The frequency tree prunes any branches with fewer than 20 protocols, prefixes longer than four gazes, or prefixes with repeated gazes. The lines of the tree are indented according to the length of the prefix. The first line representing the nil prefix includes all 960 protocols. The second line states that 785 protocols begin with a gaze on b; the line below with the same indentation states that 159 protocols begin with a gaze on ac; and the lack of any other lines with the same indentation indicates that fewer than 20 protocols began with either bd or a.

```
nil : 960 [0 ms]
   (b) : 785 [460 ms]
      (b ac) : 603 [547 ms]
         (b ac bd) : 585 [805 ms]
            (b ac bd a) : 530 [1759 ms]
      (b bd) : 181 [839 ms]
         (b bd ac) : 178 [610 ms]
            (b bd ac a) : 168 [2030 ms]
   (ac) : 159 [502 ms]
      (ac a) : 152 [563 ms]
         (ac a b) : 83 [511 ms]
            (ac a b bd) : 83 [560 ms]
         (ac a bd) : 66 [671 ms]
            (ac a bd b) : 62 [1299 ms]
```

Figure 6.10: Frequency tree for the mouse-tracking condition in the unconstrained task.

We can infer several interesting aspects of subject strategy use from the frequency tree in Figure 6.10. Subjects show a clear tendency to encode the values left-to-right: the strategy [*b ac bd a*] appears in over half the protocols. Other common strategies include: [*b bd ac a*], in which values are encoded left-to-right in *b*-*bd* and *ac*-*a* pairs; [*ac a b bd*], in which values are encoded left-to-right in pairs starting with the *ac*-*a* pair; and [*ac a bd b*], in which values are encoded in pairs starting with the larger value. The use of these four strategies represent a vast majority of the protocols (843 of 960, 88%), thus there is apparently very little deviation from these strategies. Overall the frequency tree nicely describes the pattern of strategy use in the mouse-tracking condition.

The frequency tree allows us to classify protocols as one of the four strategies to examine the adaptation of strategies over the course of the experiment. Table 6.10 shows the percentage of trials without tracing in which each strategy is used in all days and in day 1 only. (The table also includes results with tracing and two additional strategies, which will be described in later exposition.) The left-to-right strategy [*b ac bd a*] is much more prevalent in day 1 than in all days, while the other strategies are all less prevalent. Thus, subjects have a tendency to shift from the left-to-right strategy and to other strategies to decrease working memory load. As we will see in a later section, most of the difference between day 1 and subsequent days can be attributed to such a shift in three particular subjects.

Table 6.10: Mouse-tracking strategy use as percentage of total trials. Note: the percentages may not add to 100% because all trials may not correspond to identified strategies.

Strategy	Without Tracing		With Tracing	
	All Days	Day 1	All Days	Day 1
[*b ac bd a*]	55.2	64.6	60.4	69.6
[*b bd ac a*]	17.5	8.8	18.8	8.8
[*b bd a ac*]	0.0	0.0	0.2	0.8
[*ac a b bd*]	8.6	5.0	9.6	7.1
[*ac a bd b*]	6.5	2.9	10.1	11.3
[*bd b ac a*]	0.0	0.0	0.9	2.5
All	87.8	81.3	100.0	100.0

Computation. Our study of when subjects perform computation focuses on gaze durations over the equation values. The frequency tree in Figure 6.10 provides initial help in understanding subject computation through gaze durations. Looking at the branches of the tree with the above-mentioned four strategies, we see that computing gazes where values can be computed have generally longer durations than non-computing gazes. For instance, the fourth fixations are quite long for all four strategies with one exception, [*ac a b bd*]. Also, the second (non-computing) gaze for the [*b ac bd a*] takes an average of 547 ms, whereas the third (computing) gaze takes an average of 803 ms. The pattern holds strongly for [*b bd ac a*] , where the second (computing) fixation has a longer duration than the third, but holds only weakly for [*ac a b bd*] and [*ac a bd b*].

We can provide a more summative examination of computation with a statistical aggregate analysis of gaze durations over the various strategies. Every strategy of length four has the following four gazes: a start non-computing (S-NC) gaze; an intermediate non-computing (I-NC) gaze; an intermediate computing (I-C) gaze, during which computation may occur; and a final computing (F-C) gaze, during which the final computation occurs. For instance, for the left-to-right strategy [*b ac bd a*], *b* is the S-NC gaze, *ac* is the I-NC gaze (since we cannot compute knowing only *b* and *ac*), *bd* is the I-C gaze (during which *d* can be computed), and *a* is the F-C gaze. Similarly, the paired strategy [*b bd ac a*] has an I-C gaze on *bd* and a I-NC gaze on *ac*. Thus, while the S-NC and F-C gazes always appear in the strategy's first and fourth positions, respectively, the I-C and I-NC gazes may appear in either the second or third position.

By considering the protocols whose gaze prefixes match one of the four identified strategies (88% in all), we can calculate average gaze durations by strategy position without tracing. The left half of Table 6.11 shows these gaze durations. (The right half—the results with tracing—will be discussed later.) A repeated-measures ANOVA[6] shows a very significant effect on gaze duration of position, F(3,30)=14.01, p<.001. The table includes group markings that indicate specific between-position tests, where durations with different markings are significantly different, p<.05. The computing (I-C and F-C) gazes have significantly larger durations than the non-computing gazes. The F-C gaze, during which two computations can take place, has a significantly larger duration than the I-C gaze, during which only one computation can take place. The two non-computing gaze durations are also significantly different.

Table 6.11: Mouse-tracking gaze durations by strategy position, in ms. Durations with different group letter are significantly different.

Position	Without Tracing		With Tracing	
	Duration	Group	Duration	Group
S-NC	472	A	482	A
I-NC	582	B	580	B
I-C	741	C	785	C
F-C	1680	D	1661	D

Besides the overall effect of computing, it is also interesting to examine whether subjects performed interleaved computation by computing intermediate results as soon as possible—that is, during the I-C gaze. The strongest test for the presence of interleaved computation compares the durations of the I-C gazes with those of the I-NC gazes. Table 6.11 shows that these gaze durations are significantly different when aggregated across all subjects. We can also ask whether individual subjects performed interleaved computation. The left half of Table 6.12 shows the results of *t* tests that compare the subject's I-C and I-NC gaze durations for identified strategies. The first column shows the number *N* of duration data points for gazes involved in the *t* test. The second and third columns show the average gaze duration for I-NC and I-C gazes. The fourth column shows the results of the *t* test, as indicated in the table footnote. Because a large number of protocols corresponded to the strategies, there are a large

---

[6] Some ANOVA matrices used in this chapter had occasional null values in locations where no data occurred. For these cases, the null matrix values were replaced with (grand-mean + column-effect + subject-effect), computed as (row-mean + column-mean – total-mean).

number of data points for all tests. Six of the subjects show significant differences (√√) between the two positions, while two show no significant differences (“ns”). Interestingly, two subject show significant effects in the wrong direction (“??”)—that is, the I-NC gazes were significantly longer than the I-C gazes.

Table 6.12: Mouse-tracking gaze durations by subject for I-NC and I-C gazes.

Subject	Without Tracing				With Tracing			
	*N*s	Duration[a]		T-test[a]	*N*s	Duration[a]		T-test[a]
		I-NC	I-C			I-NC	I-C	
S1	68, 68	655	548	??	95, 95	630	627	ns
S2	95, 95	577	419	??	95, 95	572	429	??
S3	93, 93	784	1173	√√	96, 96	797	1166	√√
S4	47, 47	539	565	ns	87, 87	469	646	√√
S5	92, 92	682	876	√√	96, 96	689	877	√√
S6	87, 87	549	569	ns	91, 91	582	570	ns
S7	86, 86	467	962	√√	94, 94	513	1222	√√
S8	94, 94	509	685	√√	96, 96	512	681	√√
S9	96, 96	451	623	√√	96, 96	451	623	√√
S10	85, 85	605	996	√√	94, 94	578	1012	√√
All	84	582	741		94	580	785	

[a] The T-test column represents the results of a one-tailed t test as follows: √√ = significant (p<.05), √ = mildly significant (p<.1), ns = not significant (p>.1), ?? = significant in the wrong direction (p<.05).

Eye-Tracking Condition

<u>Strategy Use</u>. We now describe a similar analysis of strategy use and computation for the eye-tracking condition. Figure 6.11 shows the frequency tree for the eye-tracking condition across all days. Interestingly, this tree contains the same strategies (prefixes of length four) as the mouse-tracking condition, namely [*b ac bd a*], [*b bd ac a*], [*ac a b bd*], and [*ac a bd b*]. However, the eye-tracking tree is significantly more sparse: only 253 out of 892 protocols (28%) contain standard prefixes without repeated gazes and above the 20-protocol threshold. Also, there are a large number of prefix branches with no real regularity in strategy, such as the 311 protocols that begin [*ac bd*] but do not end without repeated gazes. Clearly there are other factors at play in these data; for instance, the 50 protocols that begin [*ac b bd*] could signify a missed initial fixation on the equation at *ac* followed by the start of the [*b bd ac a*] strategy.

```
nil : 892 [0 ms]
   (ac) : 537 [186 ms]
      (ac bd) : 311 [305 ms]
         (ac bd a) : 123 [170 ms]
         (ac bd b) : 62 [161 ms]
      (ac a) : 161 [208 ms]
         (ac a bd) : 79 [224 ms]
            (ac a bd b) : 66 [230 ms]
         (ac a b) : 57 [290 ms]
            (ac a b bd) : 53 [488 ms]
      (ac b) : 65 [175 ms]
         (ac b bd) : 51 [444 ms]
   (b) : 295 [171 ms]
      (b ac) : 218 [155 ms]
         (b ac bd) : 170 [378 ms]
            (b ac bd a) : 79 [165 ms]
         (b ac a) : 36 [213 ms]
      (b bd) : 73 [452 ms]
         (b bd ac) : 62 [212 ms]
            (b bd ac a) : 55 [547 ms]
   (bd) : 59 [255 ms]
      (bd ac) : 37 [165 ms]
         (bd ac a) : 21 [267 ms]
```

Figure 6.11: Frequency tree for the eye-tracking condition in the unconstrained task.

To emphasize this point, Table 6.13 shows subjects' strategy use as percentage of trials classified as one of the four strategies. This classification accounts for even fewer protocols in day 1 (22%) than across all days (28%). The variability and preponderance of incidental and missed fixations make it difficult to ascertain with any confidence subjects' strategy use in the task in all but a small set of protocols. As we will see, tracing can help make sense of these data and provide much more information about subjects' strategy use.

Table 6.13: Eye-tracking strategy use as percentage of total trials. Note: the percentages may not add to 100% because all trials may not correspond to identified strategies.

Strategy	Without Tracing		With Tracing	
	All Days	Day 1	All Days	Day 1
[*b ac bd a*]	8.9	9.4	15.4	14.3
[*b bd ac a*]	6.2	5.4	21.9	22.6
[*b bd a ac*]	0.0	0.0	4.6	6.0
[*ac a b bd*]	5.9	1.8	25.6	23.5
[*ac a bd b*]	7.4	5.4	21.0	20.7
[*bd b ac a*]	0.0	0.0	11.5	12.9
All	28.4	22.0	100.0	100.0

Computation. While the eye-tracking frequency tree says little about subject strategy use, it does provide some (albeit limited) insight into the effects of computation on gaze duration. Like the frequency tree for the mouse-tracking condition, the eye-tracking frequency tree shows increased duration for gazes during which computation may take place. For instance, for the strategy [*b ac bd a*], the third (computing) fixation has an average duration of 378 ms while the second fixation has an average of 155 ms. Also, for the strategy [*b bd ac a*], the second fixation (computing) fixation has an average of 452 ms while the third has an average of 212 ms. The frequency tree manifests one somewhat serious problem with the eye-tracking data: the very low gaze durations on the outer targets *b* and *a*. Qualitative analysis over many protocols from this study and the constrained study showed that subjects' gazes over the outer targets were typically very short. This problem was primarily caused by time averaging over the data, making fixations after long saccades (as the outer fixations often are) relatively short. It was also caused somewhat by parafoveal encoding of these targets. Thus, we limit our duration analysis only to gazes on the inner targets *ac* and *bd*.

The left half of Table 6.14 shows the inner gaze durations by strategy position for all protocols whose prefixes match the identified strategies (28% in all). The gaze durations overall are much smaller than those in the mouse-tracking condition; in fact, they are quite low considering that the computing gazes require either a multiplication or division. The effect of position is highly significant, F(3,30)=21.08, p<.001. However, there are no significant differences between the S-NC, I-NC, and I-C gazes. Only the F-C gaze is significantly different from the other gazes. Thus, the differences between computing and non-computing gazes is somewhat muddled in this analysis. In addition, the pattern of durations, specifically the fact that the S-NC duration is larger than the I-NC duration, does not match that in the mouse-tracking condition, raising suspicions that the data is being overwhelmed with variability.

Table 6.14: Eye-tracking gaze durations by strategy position, in ms. Durations with different group letter are significantly different.

Position	Without Tracing		With Tracing	
	Duration	Group	Duration	Group
S-NC	208	A	376	A
I-NC	165	A	443	A,B
I-C	279	A	668	B
F-C	446	B	767	B

Table 6.15 shows the results of individual subject *t* tests to check for evidence of interleaved computation on the inner targets. The *N*s are significantly smaller than those for the mouse-tracking condition; in fact, the *t* tests for four subjects were not computed because *N* was too small ($N<5$) for reliable results. As in the mouse-tracking condition, we have several subjects (S3, S7, S8, S9) that show significant effects of interleaved computing gazes. We also have one subject (S2) that shows no significant effects and one subject (S1) exhibiting significant effects in the wrong direction.

Table 6.15: Duration analysis for the eye-tracking condition with and without tracing.

Subject	Without Tracing				With Tracing			
	*N*s	Duration[a]		T-test[a b]	*N*s	Duration[a]		T-test[a b]
		I-NC	I-C			I-NC	I-C	
S1	30, 23	243	173	??	53, 11	390	1235	√√
S2	30, 14	179	127	ns	45, 5	394	125	ns
S3	11, 11	207	374	√√	41, 39	366	503	√√
S4	31, 4	-	-	-	48, 10	364	588	√
S5	16, 4	-	-	-	27, 8	295	1077	√
S6	5, 1	-	-	-	4, 1	-	-	-
S7	19, 19	146	586	√√	49, 34	686	1226	√√
S8	36, 36	222	467	√√	68, 68	327	453	√√
S9	21, 21	105	393	√√	20, 35	96	456	√√
S10	1, 1	-	-	-	38, 24	434	665	√√
All	17	165	279		31	443	668	

[a] The T-test column represents the results of a one-tailed *t* test as follows: √√ = significant (p<.05), √ = mildly significant (p<.1), ns = not significant (p>.1), ?? = significant in the wrong direction (p<.05).

[b] Durations and *t* tests are not computed if either *N*<5.

Summary

While the mouse-tracking data seem fairly robust with respect to the strategy-use and computation analyses, the eye-tracking data show significantly more variability and are more difficult to interpret. The mouse-tracking data correspond very well to the identified strategies, and although there may be less frequent and/or subtler strategies in the data, the four strategies account for almost 90% of strategy use in the experiment. The eye-tracking data provide some evidence that subjects are employing similar strategies between the conditions, but the picture is overwhelmed by the variability and noise in the protocols; this is reflected in the fact that less than 30% of the protocols correspond to identifiable strategies. The next sections describe how tracing can help clear up understanding of these data, especially for the eye-tracking condition, and can assist in analyses that incorporate a more substantial portion of the data set.

Model Development

We now describe the development of a process model grammar that captures behavior in the unconstrained task. The model will allow us to trace both the eye-tracking and mouse-tracking protocols to attempt to better understand subject behavior. Each stage of this model development (except the first exploratory stage) corresponds to a cycle of iteration as described by Ritter and Larkin (1994) comprising the steps: (1) generate model predictions, (2) trace the observed data with the model predictions, (3) analyze the fit of the model to the data, and (4) refine the model. The final product of this development is a process model grammar that describes the possible sequences of gazes. A later section extends this model grammar into a fuller ACT-R model capable of predicting the time and duration of individual gazes.

Exploratory Analysis → Model 1

The first step of model development uses exploratory analysis to derive an initial prototype model. In fact, we have already performed some exploratory analysis in the form of the frequency tree in Figure 6.11. This tree provided our first clues to the strategies employed by subjects, namely [*b ac bd a*], [*b bd ac a*], [*ac a b bd*], and [*ac a bd b*]. The fact that these strategies appeared in both the eye-tracking and mouse-tracking conditions suggests that they are indeed prevalent strategies that should be incorporated into the model. Figure 6.12 shows four sample protocols that illustrate each of the above strategies; note that while these sample protocols are good representatives of their respective strategies, many protocols embody these strategies with much more variability (e.g., incidental fixations) and/or noise (e.g., bad calibrations). The strategies appear in the frequency tree with approximately similar frequencies: [*b ac bd a*], 79; [*b bd ac a*], 55; [*ac a b bd*], 53; and [*ac a bd b*], 66.

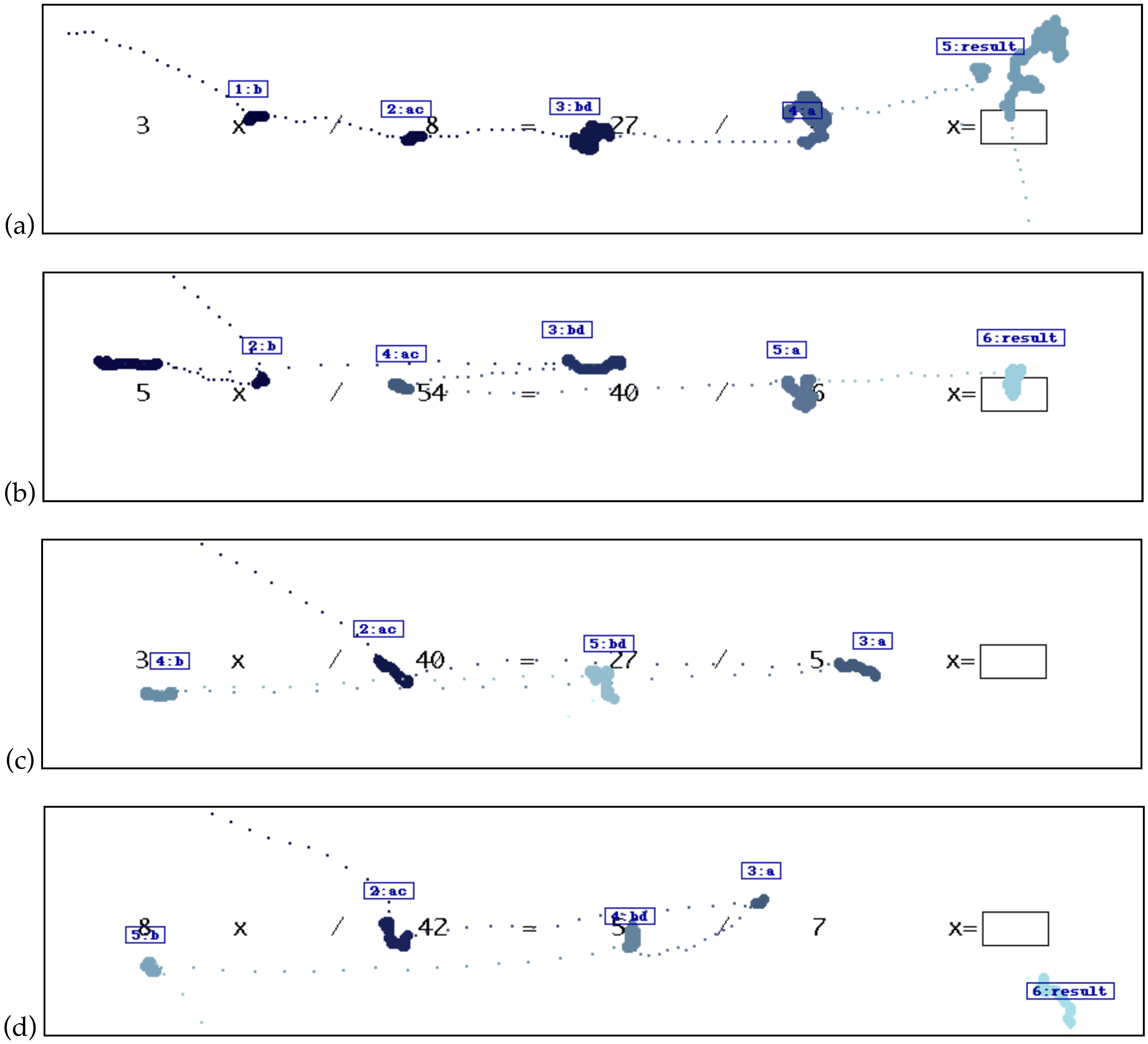


Figure 6.12: Sample protocols illustrating strategies (a) [*b ac bd a*], (b) [*b bd ac a*], (c) [*ac a b bd*], and (d) [*ac a bd b*].

These four strategies form a good basis for our first process model grammar, Model 1, shown in Table 6.16. The first rule allows the model to fixate the *start* area to begin the trial. Rules 2-5 embody the four identified strategies. Rule 6 allows for extra review fixations, similar to the model for constrained equation solving in Table 6.2. Finally, rules 7-8 either fixate the *result* area and terminate or simply terminate. The probabilities of the rules are uniformly distributed across rules for the same subgoal.

Table 6.16: Model 1 for unconstrained equation solving.

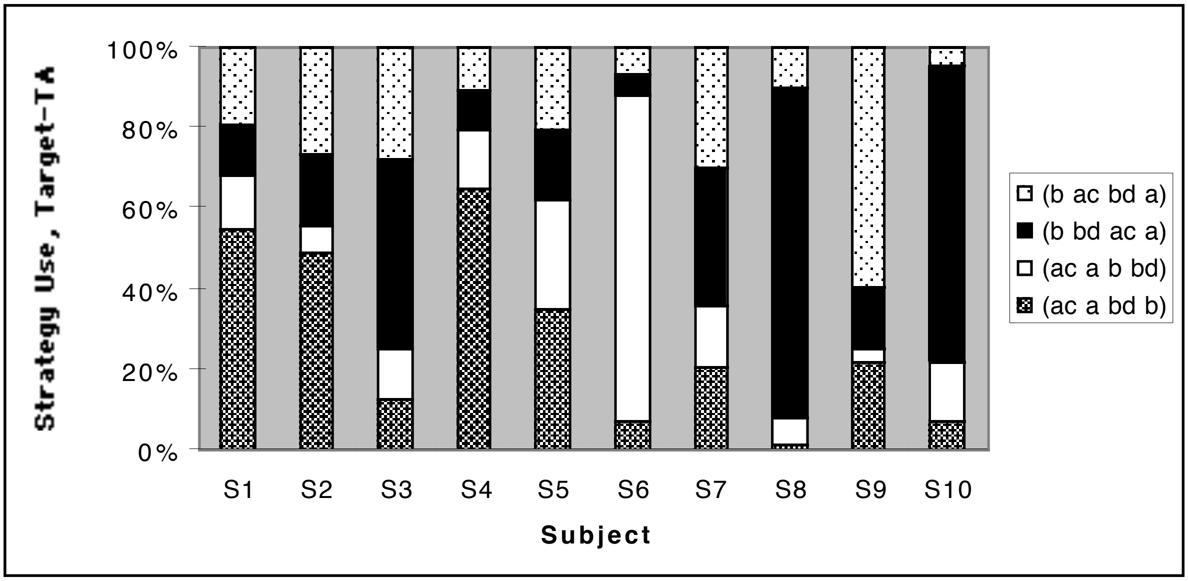
Number	Rule			Probability
1	**start-trial**	→	*start* **compute-result**	1
2	**compute-result**	→	*b ac bd a* **type-result**	1/4
3	**compute-result**	→	*b bd ac a* **type-result**	1/4
4	**compute-result**	→	*ac a b bd* **type-result**	1/4
5	**compute-result**	→	*ac a bd b* **type-result**	1/4
6	**type-result**	→	*any* **type-result**	1/3
7	**type-result**	→	*result*	1/3
8	**type-result**	→		1/3

Model 1 Analysis → Model 2

Given the Model 1 grammar, we can trace the protocol data set and analyze the results of tracing to determine where and how to refine the model. The EyeTracer system was used to trace the protocols and determine the best trace for each protocol and the resulting mismatch between model and data. The following analysis concentrates on the traces themselves; analysis of model-data mismatches arise in the following section describing the final tracing results. In addition, the analysis uses two different tracing methods, Target-TA and Fixation-HMM, to best illustrate the potential effects of tracing with different methods.

We first examine how the methods classified the protocols for each subject as one of the four Model 1 strategies. Figure 6.13 graphs the percent strategy use as interpreted by (a) Target-TA and (b) Fixation-HMM. Certain subjects are immediately salient in both graphs as utilizing a single strategy in almost all protocols, especially S6 and S8, but also S1, S2, and S10. Even when these subjects deviate from their preferred strategy, they typically employ its complementary strategy that starts with the same target. Several subjects, notably S3 and S9, show very different strategy use across the two tracing methods. Others, such as S4, S5, and S7, exhibit almost random strategy use according to the methods. There are a number of interpretations for these results: the model is perfectly classifying all protocols and subjects truly exhibit this behavior; the model is performing well on one-strategy subjects and poorly on random-strategy subjects; or the model is performing poorly on all protocols and simply forcing its interpretation onto the protocols. The only way to decide between these options is to look more deeply into the actual protocols and directly check the interpretation of these protocols given the model. However, the strategy use graphs do direct a confirmatory analysis to certain protocols that may be more problematic (e.g., the random-strategy subjects) and allow us to only briefly check protocols that seem accounted for (e.g., the one-strategy subjects).

(a)



(b)

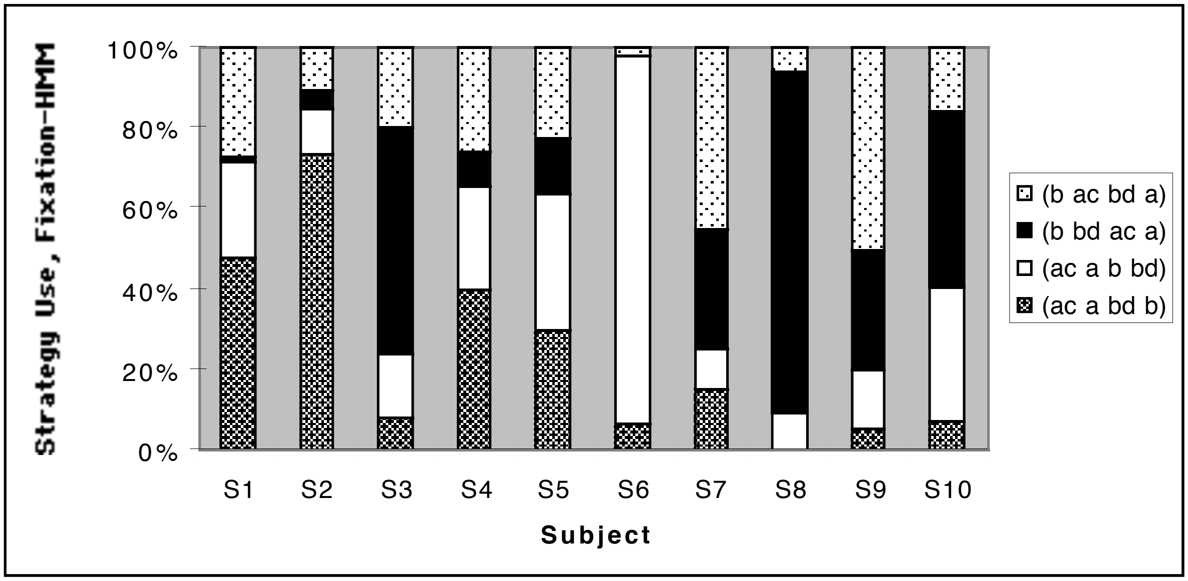


Figure 6.13: Model 1 strategy use for (a) Target-TA and (b) Fixation-HMM.

Figure 6.14 shows sample protocols from S6 and S8, two subjects whom all methods classified as using a single predominant strategy. In (a) S6 clearly utilizes the strategy [*ac a b bd*] as the tracing methods determined; note that this trace (produced by Fixation-HMM) allows for a fixation on *any* (6) after all values have been encoded. In (b) S8 employs the strategy [*b bd ac a*] and produces subsequent fixations on *any* (a review of *ac*) and *result*. These and other protocol traces from these subjects suggest that indeed the subjects utilized the strategy determined by the tracing methods.

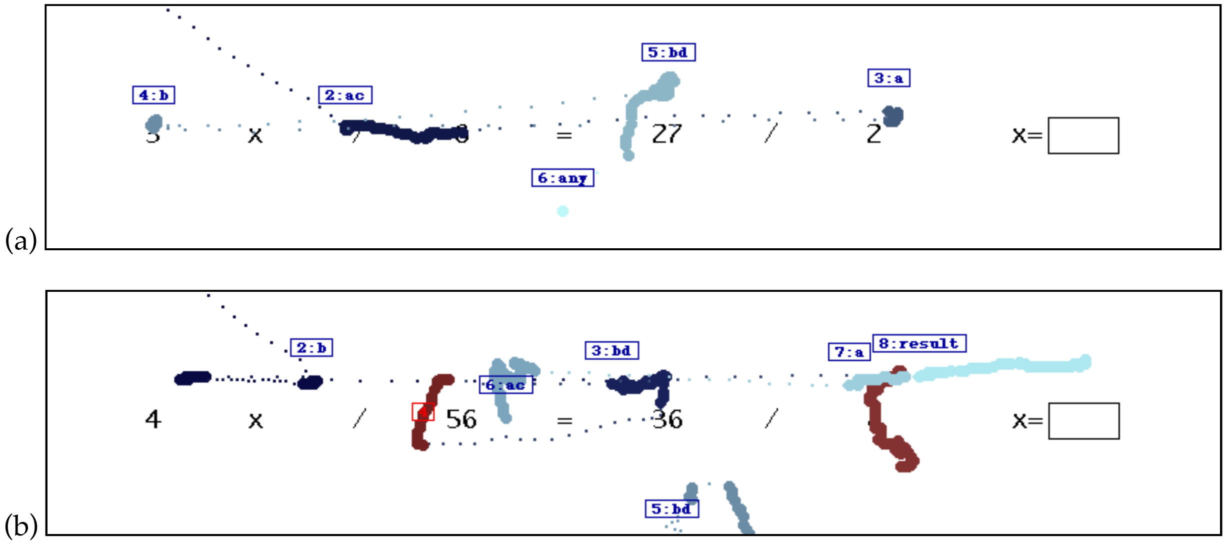


Figure 6.14: Sample protocols accurately traced by Model 1 from (a) S6 and (b) S8.

As we noted earlier, several subjects exhibited very different strategy uses as determined by the different tracing methods. To illustrate such differences, Figure 6.15 shows the same S10 protocol traced with (a) Target-TA and (b) Fixation-HMM. Because of its use of a duration threshold within a target area, Target-TA misidentifies fixation 2 as a real fixation on *ac*, though it seems more likely that these data points correspond to a saccade to *bd*. This misidentification causes Target-TA to classify the strategy as the left-to-right strategy [*b ac bd a*] instead of the more likely strategy [*b bd ac a*], thus interpreting fixation 4 as an incidental fixation. Fixation-HMM uses global sequence information to determine that this "incidental" fixation is really a saccade and interprets the protocol as the most likely strategy.

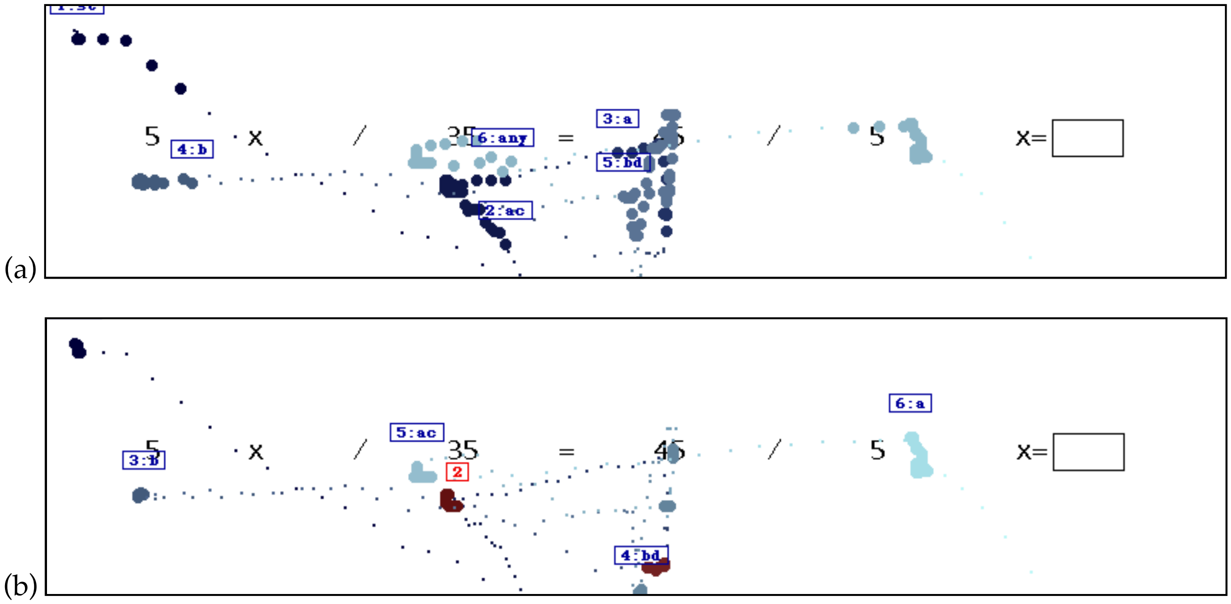


Figure 6.15: Identical S10 protocol traced with (a) Target-TA and (b) Fixation-HMM.

The subjects who exhibit “random” strategy use—i.e., none of the tracing methods finds a predominant strategy—warrant closer investigation. In looking into these subjects, we find two novel strategies not included in Model 1. Figure 6.16(a) shows a sample protocol from S5 that nicely illustrates the strategy [*bd b ac a*] (with the incidental fixation 2 on *ac*). This strategy goes against the left-to-right tendency observed earlier but produces more symmetry in encoding and computing value pairs, since the larger value is always encoded first. Similarly, Figure 6.16(b) shows a protocol from S9 that illustrates the complementary strategy [*b bd a ac*], in which the smallest value is encoded first. This protocol includes an incidental fixation on *ac* (3) and an unidentified fixation on *a*.

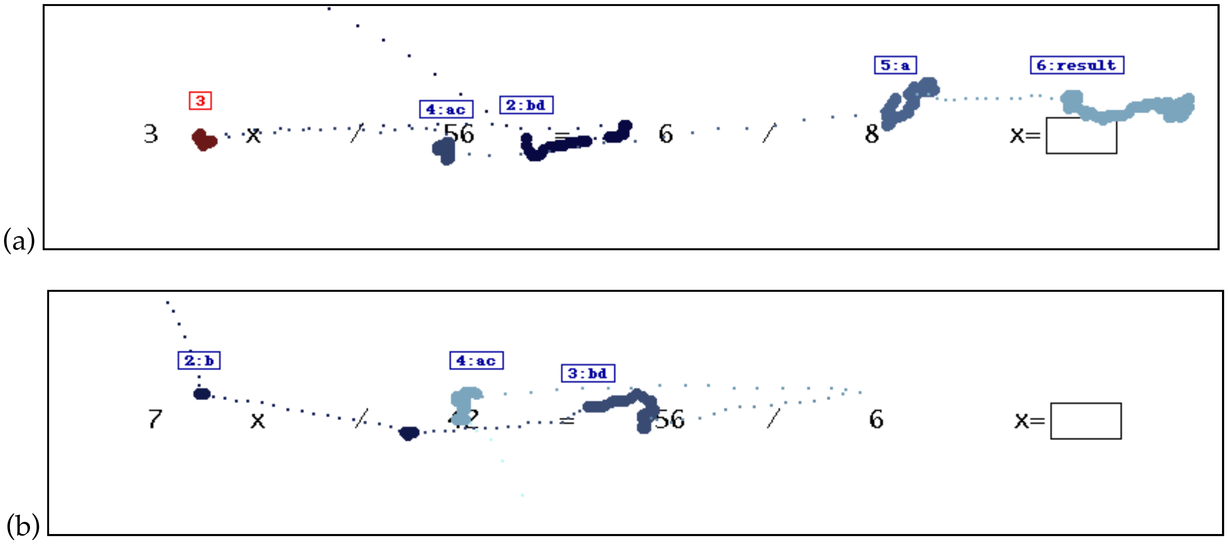


Figure 6.16: Sample protocols illustrating new strategies: (a) S5, strategy [*bd b ac a*]; and (b) S9, strategy [*b bd a ac*].

Overall Target-TA and Fixation-HMM determined that the probabilities of the various strategies are approximately uniform. Target-TA produced the following percentages for the protocols: [*b ac bd a*], 21%; [*b bd ac a*], 31%; [*ac a b bd*], 20%; and [*ac a bd b*], 28%. Fixation-HMM produced the percentages 23%, 27%, 27%, and 23%, respectively. Interestingly, the most common strategy according to Target-TA, [*b bd ac a*], was the least common according to Fixation-HMM; however, the closeness of all these values suggests that such comparisons should be taken lightly. Both methods indicate that the left-to-right strategy [*b ac bd a*] is far less prevalent in the eye-tracking condition than the mouse-tracking condition, where over half of all protocols exhibited this behavior.

This analysis of the Model 1 tracing results suggests that overall Model 1 did a good job of accounting for many of the protocols. However, our confirmatory investigation into the subjects that were not easily classified indicated two new strategies that did not appear in the frequency tree or exploratory analysis, namely [*bd b ac a*] and [*b bd a ac*]. Thus, we refine Model 1 into a new Model 2 that includes these two strategies, as shown in Table 6.17. Although the strategy [*b bd ac a*] seemed to occur more often and [*b ac bd a*] less often than the others, and the two new strategies are apparently somewhat infrequent, the analysis provided no clear reason that might justify favoring certain strategies over others. Thus, the Model 2 probabilities for the primary (**compute-result**) strategy rules remain uniform.

Table 6.17: Model 2 for unconstrained equation solving.

Number	Rule	Probability
1	**start-trial** → *start* **compute-result**	1
2	**compute-result** → *b ac bd a* **type-result**	1/6
3	**compute-result** → *b bd ac a* **type-result**	1/6
4	**compute-result** → *b bd a ac* **type-result**	1/6
5	**compute-result** → *ac a b bd* **type-result**	1/6
6	**compute-result** → *ac a bd b* **type-result**	1/6
7	**compute-result** → *bd b ac a* **type-result**	1/6
8	**type-result** → *any* **type-result**	1/3
9	**type-result** → *result*	1/3
10	**type-result** →	1/3

Model 2 Analysis → Model 3

We again analyze the fit of the model to the data in an attempt to further refine the model. Figure 6.17 shows the strategy use for subjects S1-S10 as determined by Fixation-HMM. Of the two new strategies in Model 2, patterned in diagonal lines, [*bd b ac a*] is the more dominant strategy. Comparing Figure 6.17 to Figure 6.16(b), it appears that some protocols classified as [*b bd ac a*] for Model 1 are classified as the new strategy [*b bd a ac*] for Model 2. This is likely due in large part to the fact that the strategies differ only by inversion of the final two targets. For the subjects in which the new strategy [*bd b ac a*] appears, it seems to replace classifications of protocols as [*b bd ac a*] or [*ac a bd b*]—that is, an inversion of the first two targets or the two encoding pairs, respectively. Thus the new strategies seem to help the tracing methods cope with occasional inversions of gaze orderings that appear commonly in the eye-tracking protocols. In addition, subjects that seem to vary in strategy use typically vary between two closely-related strategies—for instance, the use of S1, S2, S4, and S5 of strategies [*ac a b bd*] and [*ac a bd b*].

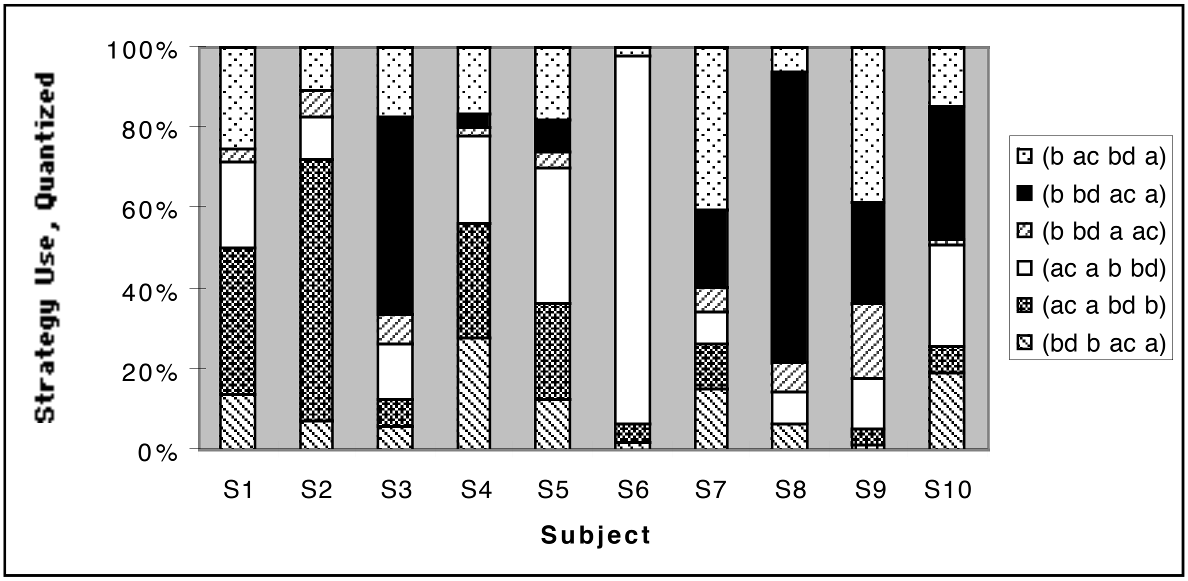


Figure 6.17: Model 2 strategy use for Fixation-HMM.

Although Model 2 is more flexible than Model 1 in allowing order inversions, several subjects still seem not to follow any particular pattern, especially S7 and S9. Figure 6.18 shows two protocols from these subjects, traced with Fixation-HMM, that illustrate some of the problems with tracing their protocols. In protocol (a), S9 exhibits a strategy very similar to the [*b bd a ac*] he exhibited in Figure 6.16(b). However, in this case, he apparently encodes *a* in the parafovea during the fixation on *bd* without actually executing a saccade to *a*. In protocol (b), S7 apparently encodes *b* during a fixation on *ac* (along with a subsequent back-and-forth motion between *ac* and *bd*). These protocols provide a good illustration of subjects' ability to encode values in the parafovea, especially the outermost values *b* and *a*, which represent a single digit in a large area of white space.

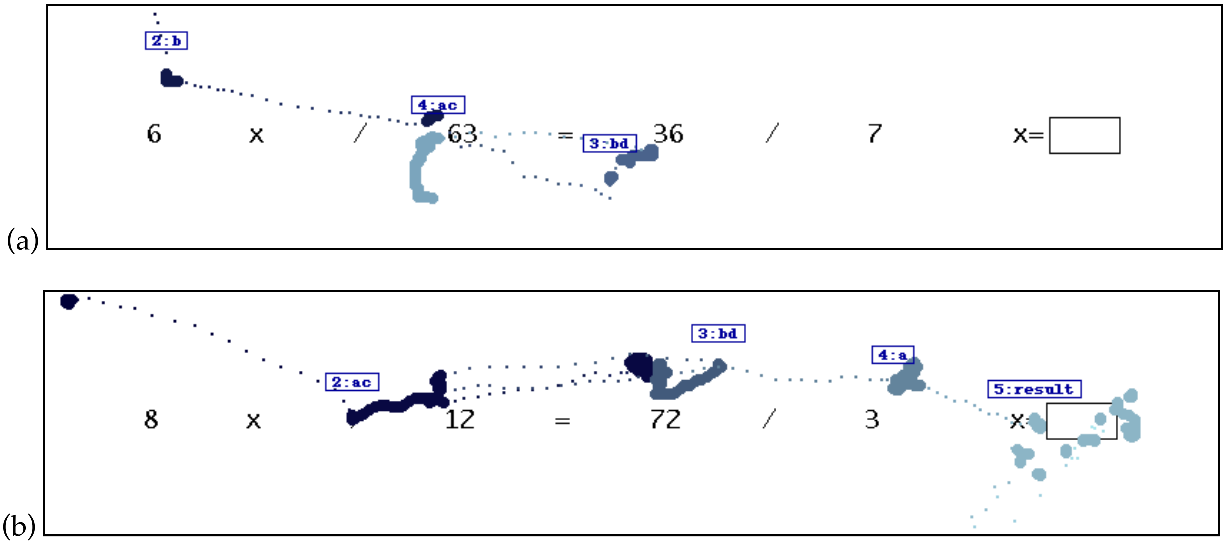


Figure 6.18: Sample protocols from (a) S9 and (b) S7.

It is clear that our process model somehow needs to capture subjects' ability to encode the outermost values parafoveally. Thus, we alter Model 2 to include strategies in which such parafoveal encoding takes place. Of the six strategies embodied in Model 2, two of them contain target pairs in which an inner target appears immediately before its adjacent outer target: [*b ac bd a*] and [*b bd a ac*]. The previous confirmatory analysis suggests that for these target pairs, subjects sometimes fixate the inner target, encode this target, then encode the outer target without moving the eye.[7] Thus, we generate new strategies for each of the two strategies by collapsing the inner-outer target pair into the inner target: [*b ac bd a*] → [*b ac bd*] and [*b bd a ac*] → [*b bd ac*]. The 1/6 probability for each non-parafoveal strategy is divided uniformly between itself and its derived parafoveal strategy. The final model, Model 3, is shown in Table 6.18.

---

[7] We might suspect that subjects sometimes do the complement: fixate the outer target, encode this target, then encode the inner target. Qualitative analysis of the protocols, however, suggested that subjects rarely if ever follow this pattern.

Table 6.18: Model 3 for unconstrained equation solving.

Number	Rule			Probability
1	**start-trial**	→	*start* **compute-result**	1
2-a	**compute-result**	→	*b ac bd a* **type-result**	1/12
2-b	**compute-result**	→	*b ac bd* **type-result**	1/12
3	**compute-result**	→	*b bd ac a* **type-result**	1/6
4-a	**compute-result**	→	*b bd a ac* **type-result**	1/12
4-b	**compute-result**	→	*b bd ac* **type-result**	1/12
5	**compute-result**	→	*ac a b bd* **type-result**	1/6
6	**compute-result**	→	*ac a bd b* **type-result**	1/6
7	**compute-result**	→	*bd b ac a* **type-result**	1/6
8	**type-result**	→	*any* **type-result**	1/3
9	**type-result**	→	*result*	1/3
10	**type-result**	→		1/3

Results with Tracing

The first attempt at analyzing the unconstrained data without tracing provided interesting results for the mouse-tracking data but mostly inconclusive results for the eye-tracking data. These results could be derived from only those protocols that matched one of the four strategies identified with the frequency trees. We now revisit these results by analyzing the protocols as they are interpreted by the process models. The eye-tracking data are traced using Model 3, which embodies both parafoveal and non-parafoveal strategies. The mouse-tracking data are traced using Model 2, since no parafoveal encoding is possible, by means of a simple sequence matching between the observed and expected mouse "gazes." The models' interpretations (i.e., protocol traces) map many more of the protocols onto identifiable strategies, thus allowing a fuller analysis of strategy use and computation.

Mouse-Tracking Condition

<u>Strategy Use</u>. The frequency-tree analysis for the mouse-tracking data was generally very effective, accounting for almost 90% of the protocols. Tracing the protocols with Model 2 effectively allows for analysis of the remaining 10%. The right half of Table 6.10 shows subjects' strategy use in the mouse-tracking condition as classified by tracing. The picture offered by tracing is overall very similar to that without tracing. Again the left-to-right strategy [*b ac bd a*] is the dominant strategy, accounting for over half the protocols. The two strategies added in Model 2, [*b bd a ac*] and [*bd b ac a*], barely appear in the mouse-tracking protocols; thus the

frequency tree, which showed only four strategies, indeed gave an accurate picture of strategy use in the condition. There is again an evident shift from the left-to-right strategy to other strategies after day 1.

To better understand the behavior and strategy shifts of individual subjects, Figure 6.19 shows individual strategy use over all four days with a 10% threshold—that is, any strategy that does not appear in at least 10% of protocols is omitted. The figure shows that most subjects focus on a single strategy throughout the four days. Half the subjects consistently use the left-to-right strategy [*b ac bd a*], one [*ac a b bd*], and one [*b bd ac a*]. All three subjects who alter strategies start with the left-to-right strategy in day 1 and eventually switch to either [*ac a bd b*] (S1) or [*b bd ac a*] (S3 and S4).

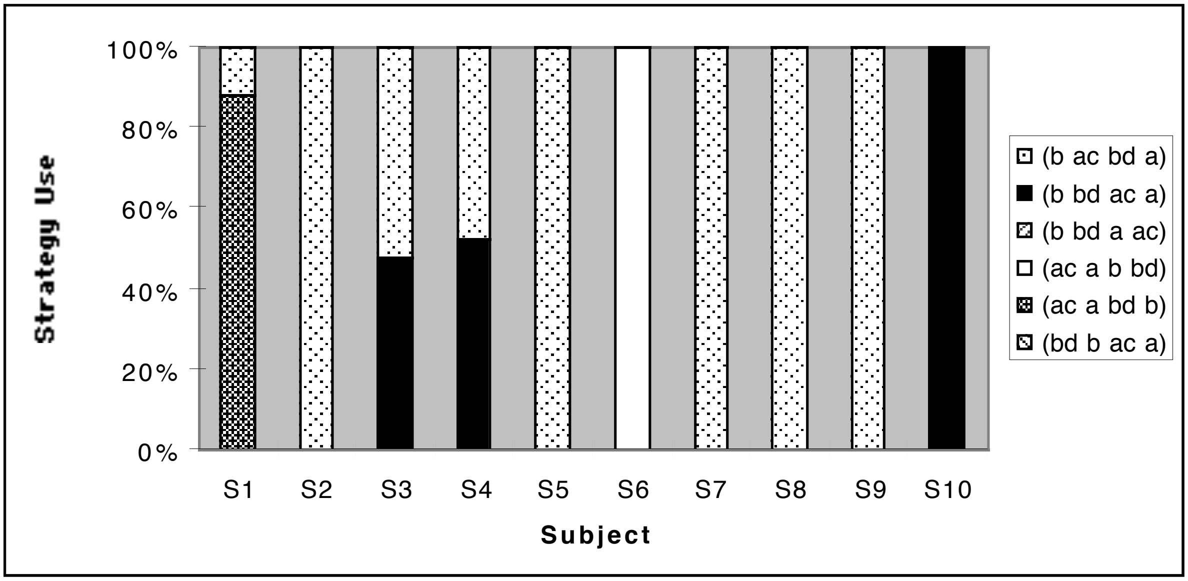


Figure 6.19: Mouse-tracking strategy use over all days, with a 10% threshold.

Tracing also allows us to determine the fit of the model to the data as the mismatch between observed and predicted sequences. Using a repeated-measures ANOVA, we can compare the mismatches in day 1 to those of days 2-4 to determine whether the model fit improved over the days. The average mismatch (number of insertions, deletions, or substitutions from one sequence to another) in day 1 was 0.38, whereas the average mismatch in days 2-4 was 0.17. However, this effect was not significant, F(1,9)=2.38, p>.1. Thus, the model fit the data approximately equally well across the four days.

Computation. Tracing also helps in a limited way to make sense when subjects performed computation. The right half of Table 6.11 shows the average gaze durations for the various strategy positions, which closely resemble those without tracing in the left half.

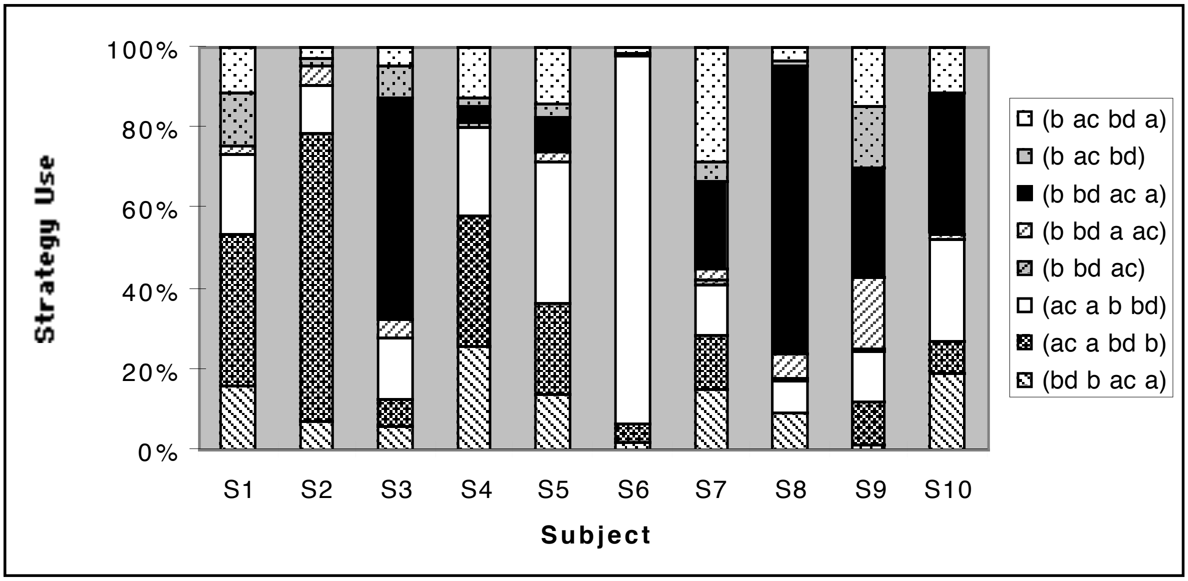
Position is again a highly significant factor, F(3,30)=16.80, p<.001. Also, the four positions are again significantly different from one another, as shown in the group markings, p<.05.

Tracing helps to clarify the picture more so with respect to interleaved computation. Without tracing, Table 6.12 shows significant effects for six subjects, insignificant effects for two, and reverse effects for two. Tracing is able to utilize almost twice as many protocols for one of the subjects, S4, turning an insignificant result into a significant one. It also makes more sense of S1's protocols and turns a reverse result into an insignificant one. In addition, tracing strengthens the size of the average effect for all subjects from 159 ms to 205 ms.

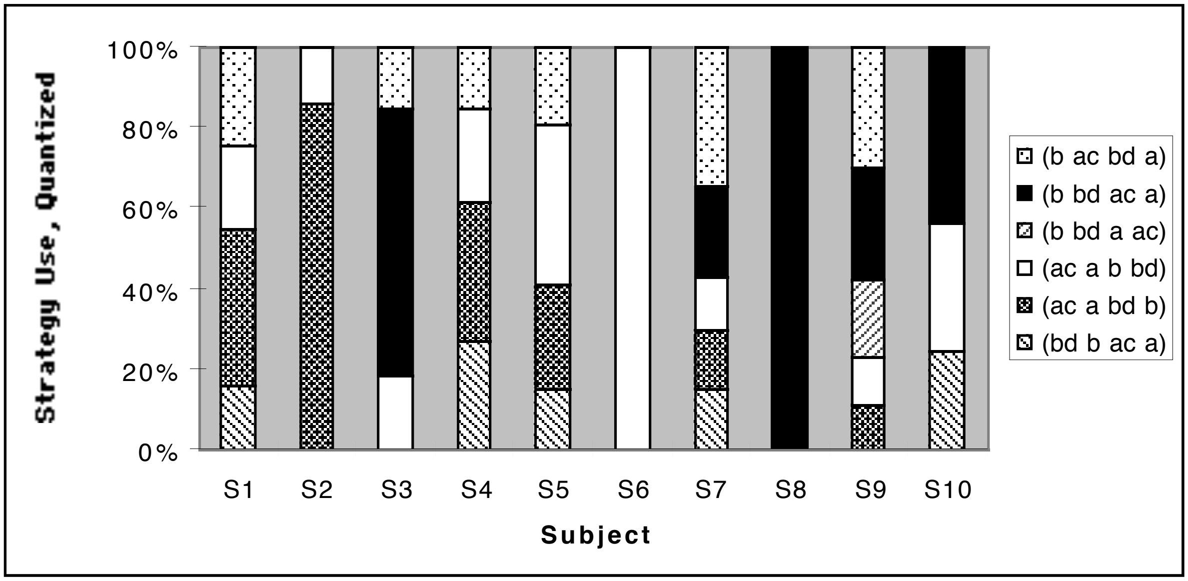
Eye-Tracking Condition

Strategy Use. The analysis of the eye-tracking data without tracing yielded generally inconclusive results in almost every respect. Tracing improves the situation significantly. Table 6.13 shows subjects' strategy use with tracing as interpreted by Fixation-HMM and Model 3. The table includes the six primary strategies, collapsing parafoveal strategies together with their non-parafoveal counterparts. The four strategies originally identified in the exploratory analysis are significantly more prevalent in the tracing analysis. However, unlike in the mouse-tracking condition, the two new strategies appear with reasonable frequency, especially [*bd b ac a*]. Strategy use is extremely similar when comparing use in all days with use in day 1 alone.

We can also examine individual subjects' strategy use for all days as shown Figure 6.20(a). The graph closely resembles that in Figure 6.17 except for the addition of the two parafoveal strategies [*b ac bd*] and [*b bd ac*]. While certain subjects exhibit extensive use of [*b ac bd*] in place of the non-parafoveal [*b ac bd a*], subjects hardly use [*b bd ac*] at all. If we collapse parafoveal strategies together with their non-parafoveal counterparts, and eliminate strategies with less than 10% frequency, subjects' strategy use looks like Figure 6.20(b).



(a)



(b)

Figure 6.20: Eye-tracking strategy use in all days, (a) complete and (b) collapsed with a 10% threshold.

It is interesting to compare subjects' strategy use in the eye-tracking condition in Figure 6.20(b) against that in the mouse-tracking condition in Figure 6.19. Some subjects clearly exhibit the same strategy use in eye-tracking as in mouse-tracking; for instance, S6 uses [*ac a b bd*] consistently in both conditions. Others exhibit eye-tracking strategy use similar to their mouse-tracking use, modulo slight variations in the strategy; for example, S1 employs [*ac a bd b*] in most trials in both conditions, but in eye-tracking sometimes employs a variant of this strategy, [*ac a b bd*] or [*bd b ac a*]. However, the majority of subjects exhibit quite different strategy use between conditions. S8 is the most salient example of these, using the left-to-right [*b ac bd a*] in all trials (above threshold) in the mouse-tracking condition but [*b bd ac a*] in the eye-tracking condition. Similarly, S2 employs [*b ac bd a*] in all mouse-tracking trials but [*ac a bd b*] in most eye-tracking trials. Overall there is a clear tendency to shift away from the left-to-right strategy to other strategies in which values are encoded in pairs. In the mouse-tracking condition, mouse movement is fairly expensive and subjects prefer to simply move left-to-right across the values, encoding them sequentially. In the eye-tracking condition, eye movement is very inexpensive, thus subjects prefer to utilize the paired strategies to reduce working memory load for storing values and intermediate results.

Although the aggregate results in Table 6.13 does not reveal much adaptation of strategy use between days, an analysis of individual subjects does reveal some, albeit limited, adaptation. Figure 21 shows subjects' strategy use in day 1 only, compared to that in all days in Figure 20. In the mouse-tracking condition, all three subjects that altered strategies changed from left-to-right to a paired strategy. Here too, three subjects showed increased use of the left-to-right [*b ac bd a*] strategy (S1, S8, and S10) in day 1 than overall. However, four subjects actually showed less left-to-right reading in day 1 than overall (S4, S5, S7, and S9). Thus, the tendency to read left-to-right is at least somewhat balanced by the ease of eye movement in the eye-tracking condition combined with the higher working memory load for left-to-right reading. With regard to model fit, a repeated-measures ANOVA reveals that mismatch between model and data decreased significantly between day 1 and days 2-4, $F(1,9)=9.51$, $p<.05$. Thus, more so than in the mouse-tracking condition, it seems that the model does not predict some initial behavior used by subjects in the course of learning.

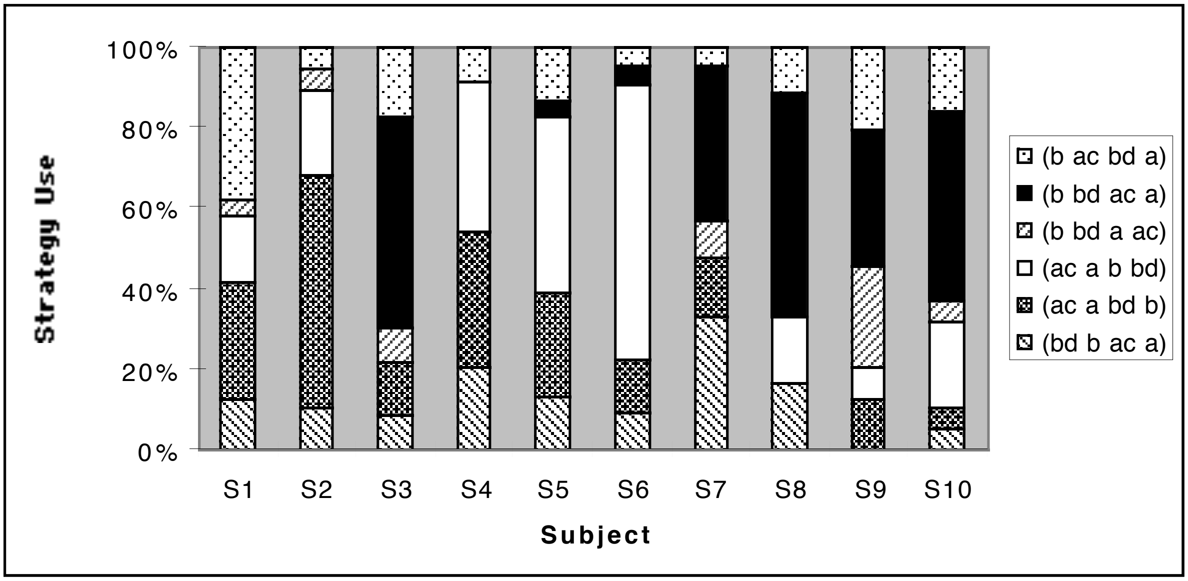


Figure 6.21: Eye-tracking strategy use in day 1, collapsed.

Computation. The average inner gaze durations by strategy position are shown in Table 6.14. The durations are much longer than those computed without tracing; this difference is likely due to the fact that the results without tracing include many incidental fixations with very small durations, while the results with tracing utilize more "real" fixations that require encoding (and possible computation) time. The position effect, as expected, is significant, F(3,30)=3.92, p<.05. The group markings show that the S-NC and I-NC gazes do not have significantly different durations, along with the I-NC, I-C, and F-C gazes. It is interesting to compare these eye-tracking results with the mouse-tracking results in Table 6.11. The mouse-tracking durations for S-NC, I-NC, and I-C are approximately 100 ms higher than their respective eye-tracking durations; the fact that subjects can preview values in the periphery in the eye-tracking condition can account for this difference. The mouse-tracking duration for F-C gazes is vastly longer than in the eye-tracking condition due to some subjects leaving the mouse over the final value while typing the result, producing very long durations for this target.

The average inner gaze durations by strategy position are shown in Table 6.14. The durations are much longer than those computed without tracing—which seems reasonable, especially for the computing gazes, given that a multiplication or division can occur. The position effect, as expected, is significant, F(3,30)=3.92, p<.05. The group markings show that the S-NC and I-NC gazes do not have significantly different durations, along with the I-NC, I-C, and F-C gazes.

The individual subject analysis of interleaved computation becomes much clearer with tracing. Table 6.15 shows how, without tracing (the left half), only six subjects could be analyzed, and one of these generated reverse results. With tracing (the right half), nine subjects could be analyzed because of the larger number of included protocols. Of these subjects, six show significant effects of computing, two show mildly significant effects, and one shows no effects. Tracing is able to produce (mildly) significant results for three subjects that could not be analyzed without tracing, namely S4, S5, and S10. It is also able to turn S1's result from a reverse result to a significant result. S6 could not be analyzed because of her consistent use of [*ac a b bd*], thus she had almost no position 2 or 3 gazes on inner targets. The eight subjects with significant effects thus seem to perform intermediate computation as soon as possible during the intermediate computing gaze. The average gaze durations are larger with tracing, averaging 668 ms for computing gazes and 443 ms for non-computing gazes.

Summary

With respect to strategy use and computation, tracing greatly facilitates the examination of the experimental data and reveals aspects of subject behavior hidden in other analyses. Especially for the eye-tracking data, tracing provided a significantly clearer picture of strategy use and adaptation, and identified strategies that the frequency trees, overwhelmed by noise, could not identify. Tracing also facilitated the analysis of computation by utilizing more interpreted protocols and bringing out or strengthening inconclusive effects.

ACT-R/PM Model

The process model grammar developed earlier provides an excellent basis for a rigorous cognitive model. In this section we examine how the model grammar can be translated to an ACT-R model (Anderson & Lebiere, 1998). There are a number of reasons to rewrite the model grammar as an ACT-R model. The model grammar dictates only possible sequences of gazes, thus omitting a number of other interesting aspects of the task. For instance, the grammar says nothing about the time at which the various gazes occur. While the strategies imply certain cognitive steps, the grammar does not make such steps explicit. In addition, the grammar does not account for other possible observed actions such as the keystrokes necessary to type a response. The following ACT-R model is not intended to represent a fully-detailed model of the equation-solving task. Rather, the model is intended to illustrate the benefits of more rigorous models and how the development of such models is a natural offshoot of the iterative trace-based analysis process.

ACT-R is a theory of cognition that posits the separation of knowledge into declarative and procedural knowledge. Declarative knowledge represents facts, or chunks, that can be explicitly stated, such as the fact that 2 + 2 = 4 or that Harrisburg is the capital of Pennsylvania. Procedural knowledge represents the production rules needed to retrieve declarative knowledge and act upon it, but the productions themselves cannot be explicitly recalled or stated. The ACT-R theory has proven successful at being able to model a wide variety of problem-solving domains, including list memory, choice, analogy, and scientific discovery (see Anderson & Lebiere, 1998). When augmented with its perceptual-motor interface, ACT-R/PM (refs), as it is called, can interact with the outside world by encoding on-screen visual elements and executing motor commands.

Model Overview

An ACT-R model is specified by its declarative knowledge in the form of chunks and its procedural knowledge in the form of productions. The following equation-solving ACT-R/PM model is based on the Model 2 grammar.[8] To avoid a lengthy exposition on the details of the model, this description provides only an overview of the declarative and procedural knowledge needed. Interested readers can find the entire model, along with the LISP code needed to run simulations of the model, at this paper's Web site.[9]

The model's declarative knowledge includes two types of chunks: multiplication facts and value positions. The multiplication facts represent the knowledge of integer products, used by productions for both multiplication and division. For instance, the chunk THREE*FOUR would encode the fact that 3 * 4 = 12. Value positions give a visual location for each value in the equation, namely b, ac, bd, and a. The visual productions use this information to find and encode the values when needed.

The model's procedural knowledge comprises productions that embody a number of possible subgoals. The top-level goal *solve-problem* subgoals one of the six possible strategies in the Model 2 grammar. Each strategy is itself a separate subgoal with several productions for encoding the equation values and computing the values of c and d. The strategies assume that intermediate values are computed as soon as possible. To encode values, the strategies set the subgoal *encode-value* to visually locate the desired item and return its value to its parent goal. To compute results, the productions retrieve the appropriate multiplication fact for the given two elements to find the desired third element. After the subgoaled strategy has computed c and d, the *solve-problem* goal computes the final result and sets a subgoal to type this result. The final result is typed by one of two productions, one of which types a one-digit value and the other which types a two-digit value. Finally, the *solve-problem* goal types a return character to

---

[8] ACT-R/PM does not actually make predictions of where the eye moves, only of where visual attention moves. Thus, at this time, it has only limited ability to model parafoveal encoding, preventing us from basing the equation-solving model on the Model 3 grammar.

[9] http://www.cs.cmu.edu/~dario/TH99/

end the trial. Visual attention is assumed to shift to the result box when the first digit is typed to the box.

This model interfaces with a simulated experiment that runs in the LISP environment. The experiment puts up a window with the given equation and provides an editable text box in which to type a response. When run with a human subject, the human can simply read and type into the window normally. When run with ACT-R/PM, the model's encoding and typing actions are translated to and executed in this window. Thus ACT-R/PM allows both human and model to interface with the same experiment window.

Results

To illustrate the benefits of the ACT-R model, we focus our results analysis on only one aspect of task behavior: gaze durations on the various strategy positions. A single parameter, the base activation of all chunks, was estimated to have the value 1.5. Table 6.19 shows the average gaze durations by strategy position for the eye-tracking data and the ACT-R model. The model provides a good fit to the data in most respects, $R = 0.97$. The two computing positions I-C and F-C have significantly longer durations than the non-computing positions. Also, the F-C position, which can account for two computations, has a longer duration than the I-C position, which can account for only one computation. Unlike the data, the model shows no difference between the S-NC and I-NC positions.

Table 6.19: Gaze durations for the eye-tracking data and the ACT-R/PM model, in ms.

	S-NC	I-NC	I-C	F-C
Data	376	443	668	767
Model	338	338	611	881

Clearly, the equation-solving model as described is deficient on a number of counts. First, it does not account for the probabilities of the various strategies, though we could add the probabilities observed in the data to the model in a straightforward manner. Second, the model fails to exhibit any of the great variability the subjects exhibited. Third, it does not exhibit the slight adaptations in strategy use displayed by subjects. Further refinement of the model to alleviate these deficiencies is beyond the scope of this chapter. Nevertheless, we have every reason to believe that tracing, by facilitating understanding of subject behavior, can assist in refinement of the model with respect to all of these issues.

The fact that the model fits the observed gaze durations so well is a nice indication that the model contributes to our understanding of behavior in the equation-solving task. Since the Model 2 grammar only predicts the sequential order of gazes, there is no a priori reason for the gaze durations to be predicted accurately. The good model fit illustrates that, given a simple implementation of the identified gaze sequences, ACT-R can accurately predict the amount of time needed to encode the equation values and to compute intermediate and final results. This model, unlike the Model 2 grammar, reveals subject behavior at a much smaller grain size than individual gazes, since it predicts the cognitive steps that direct this behavior in addition to other motor actions (i.e., typing) that are included the behavior. Thus, this simple model fit is a proof of concept that translating a simple model grammar into a more rigorous theoretical model can provide additional insight into behavior not available in the grammar's gaze sequences alone.

The ACT-R model and, more generally, much of the unconstrained equation-solving study manifest an important aspect of the tracing methods: their ability to sort out temporal aspects of behavior using the sequential aspects of that behavior. Tracing, as described in this thesis, uses only sequential information to interpret observed protocols. This study shows that by tracing the protocols with sequential information, the temporal information in the protocols (e.g., gaze durations that indicate computation) becomes more clear and understandable. The interaction between sequential and temporal information is the primary focus of the next chapter on reading, a domain in which this interaction is crucial to understanding behavior.

Chapter 7

Reading

Introduction

Of all the higher-level domains in which researchers have studied eye movements, reading has enjoyed the greatest attention. Although readers often have the impression that their eyes sweep across text in a steady motion, their eyes in fact produce rapid saccades and fixations in subtle and interesting patterns. Researchers have typically studied eye movements in reading by aggregating them into fixations or gazes and analyzing these aggregate units (e.g., Just & Carpenter, 1980). Such studies have led to numerous findings on the relation between cognition and eye movements in reading, such as where the eyes land (e.g., O'Regan, 1981), how much the eyes can see and encode (e.g., McConkie & Rayner, 1975), and how characteristics of the text affect reading performance (e.g., Schilling, Rayner, & Chumbley, 1998). Several researchers have developed theories of reading based on these findings and instantiated them as rigorous computational models (e.g., Just & Carpenter, 1980; Legge, Klitz, & Tjan, 1997; Morrison, 1984; O'Regan & Levy-Schoen, 1987; Reichle, Pollatsek, Fisher, & Rayner, 1998).

The study and modeling of eye movements in reading typically address two types of information in the data: temporal information in the form of fixation durations, and spatial information in the form of fixation/gaze locations. While these types of information provide a good picture of reading behavior, research in reading has paid less attention to another important aspect of eye movements: sequential information. It may seem surprising at first that sequential information is an important component of eye movements in reading; one might have the impression that the eye simply moves left to right with few interesting deviations. On the contrary, this sequential information can be quite complex. When considering only left-to-right eye movements at the word level, the eye sometimes re-fixates words or skips some words entirely. These possibilities produce interesting transitions where, from a given word, the eye may move to the same word, the next word, or any subsequent word with some likelihood. In addition, not all reading eye movements are left-to-right; approximately 10% of saccades during reading are regressions, or backward saccades. Thus, sequential information in various forms can nicely complement temporal and spatial information and provide a fuller picture of human reading behavior.

This chapter describes a study of how tracing facilitates the evaluation of computational models of reading and helps sort out temporal information in reading data given sequential information. Whereas the equation-solving study developed and refined a cognitive model from scratch, the reading study compares two existing models of reading, E-Z Readers 3 and 5 (Reichle, Pollatsek, Fisher, & Rayner, 1998), and shows the benefits of tracing for such existing models. E-Z Readers 3 and 5 represent two of five E-Z Reader models developed to predict eye movements in reading at the word level. Reichle et al. fit all five E-Z Reader models to several aspects of a previously-collected data set (Schilling, Rayner, & Chumbley, 1998), specifically to fixation durations and probabilities on words of different frequencies. E-Z Readers 3 and 5 were found to provide the best fits with respect to these measures. However, because both models provided equally good quantitative fits to these measures, Reichle et al. could rely only on qualitative examination to discern which model was indeed the more accurate model of reading behavior.

The reading study employs the E-Z Reader models in two parts. The first part of the study compares E-Z Readers 3 and 5 with respect to sequential information in an attempt to provide quantitative support for favoring one of the models. The comparison utilizes a newly-collected data set based on the original data that, unlike the original, includes raw eye-movement data that can be traced with the tracing methods. The study first derives first-order and second-order model grammars for each model—that is, a grammar that incorporates the probabilities of the E-Z Reader models' first- and second-order transitions. It then traces the data set protocols with the model grammars, producing goodness-of-fit measures between each model and the data. These measures allow for a quantitative comparison of the models with respect to sequential information where comparisons of other aspects are inconclusive.

The second part of the study uses the sequential information in the E-Z Reader models to sort out temporal aspects of the data. Compared to the original data, the new data was collected with a less accurate eye tracker at a coarser sampling rate. When analyzed in the standard way by mapping fixations to their nearest target, certain aspects of the data are less clear and conform neither with the original data nor with what we would reasonably expect. When analyzed using the tracing methods and the E-Z Reader models, these aspects of the data become clearer and provide a better match to expected values. The reading study, like the equation solving study in the last chapter, thus shows how tracing facilitates understanding of temporal information given sequential model predictions.

Both parts of the study utilize first- and second-order model grammars derived from the original E-Z Reader models. The use of these grammars brings up an interesting difference between the reading study and the equation-solving studies. For equation solving, the number of potential strategies that subjects could employ was relatively low, thus allowing us to enumerate every strategy fully in the model grammar. For reading, the number of potential strategies—that is, possible sequences of word fixations—can be extremely high; for instance, for left-to-right strategies on a 10-word sentence in which each word may be fixated or skipped, there may be up to $2^{10} = 1024$ strategies. It is generally infeasible to enumerate this many strategies fully in a model grammar. Thus, we must reduce the full grammar to a more feasible grammar that, though it contains less information, incorporates as much of the important aspects of the original grammar as possible. The reading domain and the eye-typing domain in the next chapter provide good illustrations of how this reduction can be realized.

Because of the nature of the first- and second-order model grammars, the reading study utilizes only the HMM-based tracing methods, namely fixation tracing and point tracing. Target tracing, due to its basis in sequence matching, requires that grammars be expanded into lists of every possible strategy; even with a threshold on the number of cycles or length of strategies, such a process would generate an infeasible strategy list for the first- and second-order reading grammars. Thus, while target tracing provided good if not excellent results in the equation-solving studies, we cannot employ it at all in the reading study. This study, like the eye-typing study, illustrates how the novel HMM-based tracing methods facilitate analysis even for domains in which sequence-matching methods are entirely inapplicable.

E-Z Reader

We begin our reading study with an outline of the E-Z Reader class of models and, in particular, E-Z Readers 3 and 5. The E-Z Reader class of models (Reichle, Pollatsek, Fisher, & Rayner, 1998) represent five computational process models of eye-movement control in reading. The models emphasize the relation between eye movements and cognition, particularly lexical access, by predicting the duration and number of fixations on words of various frequencies. Reichle et al. showed that these models provide an excellent fit to word-frequency effects in data collected by Schilling, Rayner, and Chumbley (1998). In particular, they showed that E-Z Readers 3 and 5 generated especially good fits and should be considered the most faithful of the E-Z Reader models. The following sections outline these two models and provide details necessary for understanding the remainder of the chapter. Interested readers can consult

Reichle et al. (1998) or Rayner, Reichle, & Pollatsek (1998) for detailed information about these and the other E-Z Reader models.

E-Z Reader 3

E-Z Reader 3 comprises a number of component processes and subprocesses: lexical access, with a familiarity check f and a complete lexical access lc; inter-word saccade programming, with a labile (retractable) stage m and a non-labile (unretractable) stage M; intra-word saccade programming, with a labile stage r and a non-labile stage R; and the actual saccadic eye movement. In this exposition we first examine each of these processes in detail. We then analyze the control flow of the model and how the various processes come together as a full model of eye-movement control in reading.

Lexical Access

Lexical access is the central process in E-Z Reader 3 that drives when saccade programming, and thus eye movement, takes place. Lexical access can be broken into two subprocesses: a familiarity check f and a complete lexical access lc. The familiarity check f determines when the currently-attended word is recognized as being familiar—that is, the point at which complete lexical access is imminent. The complete lexical access lc completes access of the word. The division of these two subprocesses forms a distinction between matching on the basis of global similarity and retrieval of the actual memory item, a distinction shown in other models of memory (see Reichle et al., 1998). As we will see, the distinction helps to explain saccadic programming in that a saccade is programmed to the next word as soon as the familiarity check on the current word ends.

The familiarity check f_n is initiated when a new word n is attended. The time needed for familiarity check depends on two properties of word n: its frequency $freq_n$ as the frequency with which the word appears in standard text (Francis & Kucera, 1982); and its predictability p_n as the probability with which the word can be predicted based on previous words. Based on these two properties, the time $T(f_n)$ for the familiarity check f on the word n is assumed to be distributed as a gamma distribution with mean $t(f_n)$ and standard deviation $t(f_n)/3$, where $t(f_n)$ is defined as:

$$t(f_n) = \left[f_b - \left[f_m * \ln(freq_n)\right]\right] * (1 - \theta * p_n) \qquad \text{(Equation 1)}$$

The parameters f_b and f_m are the intercept and slope parameters, respectively. The parameter θ defines the attenuation factor that reduces the effect of predictability on the familiarity check. When f_n ends, the model initiates the completion of lexical access lc_n.

The complete lexical access lc_n represents the actual access of word n in memory. Like f_n, lc_n depends on the frequency and predictability of word n. The time $T(lc_n)$ for lc_n is distributed as a gamma distribution with mean $t(lc_n)$ and standard deviation $t(lc_n)/3$, where $t(lc_n)$ is defined to be:

$$t(lc_n) = \Delta * \left[f_b - \left[f_m * \ln(freq_n)\right]\right] * (1 - p_n) \qquad \text{(Equation 2)}$$

Here there is no attenuation parameter θ and the factor Δ scales the effect of frequency. When lc_n ends, the model shifts attention to word n+1 and initiates f_{n+1} on this word.

Saccade Programming and Execution

E-Z Reader 3 includes two types of saccade programs, one for inter-word saccades and one for intra-word saccades. Each type of saccade program has two stages: a labile, or retractable, stage that can be canceled under certain conditions; and a non-labile, or non-retractable, stage that cannot be canceled and executes a saccade upon completion. We call the labile and non-labile stages for inter-word saccades m and M, respectively, and for intra-word saccades r and R, respectively. It is important to note that the processes of lexical access and saccade programming and execution can be interleaved, producing a dissociation between attention and fixation location. This dissociation is essential to modeling aspects of real-world reading behavior such as skipping over high-frequency words, since these words can be accessed and recognized quickly in the parafovea without being fixated.

For inter-word saccades, E-Z Reader 3 programs an inter-word saccade to word n+1 after the familiarity check of word n has ended. Formally, upon completion of f_n, the model initiates the labile stage m_{n+1} in addition to the lexical access lc_n. Normally when this stage completes, the model simply initiates the non-labile stage M_{n+1} and executes a saccade to the next word upon completion of M_{n+1}. However, it is possible that while m_{n+1} runs, the model completes both lc_n and f_{n+1}; this is especially likely if word n and/or n+1 is a high-frequency or high-predictability word. In this case, the completion of f_{n+1} cancels the previous program m_{n+1} and initiates an inter-word saccade for the subsequent word, m_{n+2}. This sequences of events results in the model skipping (i.e. not fixating) word n+1, producing a saccade from word n directly to word n+2 (unless that word is skipped, and so on). Note that the non-labile stage M_{n+1} can never be canceled and always produces a saccade upon completion.

For intra-word saccades, the model in most cases programs an intra-word saccade to the currently-attended word upon completion of either an inter- or intra-word saccade; specifically, it initiates the labile stage r_n to the currently-attended word n when any M or R completes, provided there are no pending m or M processes. As for inter-word saccades, completion of the labile stage r_n initiates the non-labile stage R_n after which a saccade to word n occurs. However, completion of f_n before r_n cancels r_n, since the model starts an inter-word saccade program at that time. Also, because r and R always follow the currently-attended word, a switch of attention from word n to word n+1 (from a completion of lc_n and initiation of f_{n+1}) cancels any existing r_n or R_n and initiates a corresponding r_{n+1} or R_{n+1}. Because the currently-attended word can actually be a word previous to the currently-fixated word, intra-word saccades can occasionally produce regressions (saccades to previous words).

The times needed to run all four processes, like those of lexical access, are assumed to have a gamma distributions with standard deviations equal to one third the means. The times $T(m_n)$ and $T(r_n)$ for the labile stages are set to have the same means $t(m_n) = t(r_n)$ and variances $t(m_n)/3 = t(r_n)/3$. Likewise, the times $T(M_n)$, and $T(R_n)$ for the non-labile stages have the same means $t(M_n) = t(R_n)$ and variances $t(M_n)/3 = t(R_n)/3$.

Parameter Settings

E-Z Reader 3 requires settings for six model parameters and two word parameters. The three model parameters f_b, f_m, and Δ are estimated to maximize the model-data fit, as we discuss in the upcoming study. The remaining three model parameters are preset as follows: mean duration of labile saccade programming $t(m_n) = t(r_n) = 150$ ms; mean duration of non-labile saccade programming $t(M_n) = t(R_n) = 50$ ms; and the familiarity check parameter $\theta = .5$. The two word parameters, frequency $freq_n$ and predictability p_n, are taken from the standard frequency norms (Francis & Kucera, 1982) and an empirical study (Reichle et al., 1998), respectively. Finally, the model assumes that an executed saccade has a duration of 25 ms.

Control Flow

The control flow of E-Z Reader 3 can be described as a queue of processes where the queued processes run in parallel and start other processes upon completion.[10] The model first initializes the queue with two running processes, f_0 and $r_.$ Then, at each iteration, the model sorts the processes in the queue with respect to completion time and removes the process with the soonest completion time. The completion of this process triggers some number of actions, as described in Table 7.1; the actions can include starting a new process (i.e., adding it to the queue), canceling a process (i.e., removing it from the queue), or executing a saccade. The completed process actions in the table conform to the above descriptions of process initiations and cancellations for lexical access and saccade programming and execution.

---

[10] This representation is equivalent to the Order of Processing (OP) diagrams used by Reichle et al. (1998) but is somewhat more compact and may help to clarify certain transitions not explicitly included in their OP diagrams. Please refer to Reichle et al. (1998) for a full description of the more graphical representation embodied by the OP diagrams.

Table 7.1: E-Z Reader 3 control flow showing actions for each process upon completion.

Process	Actions upon Completion
f_n	If m_n is pending Then Cancel m_n Start lc_n Start m_{n+1} Cancel r_n
lc_n	If r_n is pending Then Cancel r_n and Start r_{n+1} If R_n is pending Then Cancel R_n and Start R_{n+1} Start f_{n+1}
m_n	Start M_n
r_n	Start R_n
M_n	Execute saccade to word n If no R or M is pending and the queue is not empty Then If f_{n-1} or lc_{n-1} is pending Then Start r_{n-1} Else Start r_n
R_n	Execute saccade to word n If no R or M is pending and the queue is not empty Then Start r_n

To better illustrate the control flow of E-Z Reader 3, Table 7.2 shows a sample sentence with fixations generated by a single model simulation run. We see that the model fixates only particular words, namely words 0, 1, 2, 5, 6, and 8. The model fixates word 6 twice, while skipping over words 3, 4, and 7 entirely. In general, the skipped words are more frequent words and thus require less time for both subprocesses of lexical access. The durations of fixations prior to skipped words are generally longer, since they include the time needed to encode (i.e., perform lexical access) the word on which they land in addition to subsequent skipped words. Of course, because of the probabilistic nature of the model, we can expect potentially very different fixation sequences on different simulation runs.

Table 7.2: Fixations for a sample sentence from one E-Z Reader 3 simulation run.

Word	0	1	2	3	4	5	6	7	8
Sentence	The	policeman	demanded	to	see	Jim's	license	and	registration.
Fixation Duration	279	168	468			139, 229	322		227

Table 7.3 shows a partial trace of the fixation sequence in Table 7.2, including each step of the model, the queued processes during that step, and the current fixations at the step. Each queued process is shown with its time of completion (in ms). As always, the model starts the queue with f_0 and r_0 and sets the currently-fixated word to word 0 (step 1). Because f_0 is the first queued process to end, the model removes it from the queue and starts the lc_0 process (step 2); also, it cancels r_0 and starts m_1 to fixate the next word. The model continues and processes words 1 and 2 (steps 3-12); note that the current fixation changes 25 ms after final saccade programming M completes, given that 25 ms is the assumed time needed for a saccade. When the model reaches word 3 (step 13), it completes f_3 before m_3, thus it cancels m_3 and starts m_4 to fixation word 4 while lc_3 runs (step 14). A similar situation occurs for word 4 (steps 15-16), after which the model completes saccade programming to word 5 (steps 17-18) and fixates word 5 (step 20). For word 5, r_5 ends before f_5 (step 19), causing the model to queue R_5 (step 21) and eventually re-fixate word 5 (step 24). The model then fixates word 6, skips word 7, and fixates word 8 in a similar manner (steps 25 and beyond).

Table 7.3: Partial trace of the simulation run in Table 7.2.

Step	Time	Queued Processes	Fixation
1	0 ms	$(f_0\,, 26)$ $(r_0\,, 162)$	Word 0
2	26	$(lc_0\,, 73)$ $(m_1\,, 186)$	
3	73	$(m_1\,, 186)$ $(f_1\,, 337)$	
4	186	$(M_1\,, 254)$ $(f_1\,, 337)$	
5	254	$(f_1\,, 337)$ $(r_1\,, 389)$	
6	279		Word 1
7	337	$(m_2\,, 388)$ $(lc_1\,, 436)$	
8	388	$(M_2\,, 422)$ $(lc_1\,, 436)$	
9	422	$(lc_1\,, 436)$ $(r_1\,, 587)$	
10	436	$(f_2\,, 590)$ $(r_2\,, 707)$	
11	447		Word 2
12	590	$(lc_2\,, 655)$ $(m_3\,, 724)$	
13	655	$(f_3\,, 688)$ $(m_3\,, 724)$	
14	688	$(lc_3\,, 724)$ $(m_4\,, 850)$	
15	724	$(f_4\,, 775)$ $(m_4\,, 850)$	
16	775	$(lc_4\,, 808)$ $(m_5\,, 872)$	
17	808	$(m_5\,, 872)$ $(f_5\,, 1047)$	
18	872	$(M_5\,, 890)$ $(f_5\,, 1047)$	
19	890	$(r_5\,, 996)$ $(f_5\,, 1047)$	
20	915		Word 5
21	996	$(R_5\,, 1029)$ $(f_5\,, 1047)$	
22	1029	$(f_5\,, 1047)$ $(r_5\,, 1181)$	
23	1047	$(m_6\,, 1203)$ $(lc_5\,, 1236)$	
24	1054		Word 5
…	…	…	…

E-Z Reader 5

E-Z Reader 5 is identical to E-Z Reader 3 in all respects but one: the addition of two *eccentricity* parameters. E-Z Reader 3 assumes that attention to any word in the given sentence has equal efficacy—that is, it considers parafoveal viewing to be equivalent to foveal viewing. To model the increased difficulty of viewing words in the parafovea, E-Z Reader 5 incorporates two eccentricity parameters ε_1 and ε_2 that represent the attenuation of viewing ability in the parafovea. These parameters are included in the duration equations of familiarity check f_n and lexical access lc_n as

$$t(f_n) = \left[f_b - \left[f_m * \ln(freq_n)\right]\right] * (1 - \theta * p_n) * \varepsilon_1^{\,x} \qquad \text{(Equation 3)}$$

$$t(lc_n) = \Delta * \left[f_b - \left[f_m * \ln(freq_n)\right]\right] * (1 - p_n) * \varepsilon_2^{\,x} \qquad \text{(Equation 4)}$$

where x is the distance of the currently-attended word to the currently-fixated word (as the number of words). Thus both durations become longer when the currently-attended word is farther from the currently-fixated word—that is, farther from the fovea.

Reading Study

The reading study demonstrates how tracing helps compare computational models of reading with respect to sequential information and helps make more sense of temporal and spatial information given this sequential information. The study applies first- and second-order model grammars derived from E-Z Readers 3 and 5 to a newly-collected data set based on an earlier data set collected by Schilling, Rayner, and Chumbley (1998). In this experiment, which we will term the "SRC" experiment (after the authors), subjects read medium-length sentences and occasionally answered questions to ensure that they had processed the information. Reichle et al. (1998) analyzed these data to manifest effects of word frequency on fixation durations and probabilities. Their overall results, like those of Schilling et al. (1998) show that less frequent words (i.e., words that appear less frequently in standard usage according to Francis & Kucera, 1982) required longer fixations and more fixations on average than more frequent words.

This study of reading replicates the SRC experiment and uses E-Z Readers 3 and 5 to trace and analyze the data. Although the original data were available for use, they included only duration, word position, and letter position of each fixation; they did not include the millisecond-by-millisecond raw data required for the tracing algorithms. The experiment in this study, which we term the "New" experiment, gathers these raw data and, at the same time, replicates the experimental results with a different eye tracker and subject population. The eye tracker used in the New experiment was, unfortunately, somewhat less accurate than that used in the SRC experiment (accuracies of 1/2-1° for New versus 1/6° for SRC). However, as we will see, tracing helps make sense of the less accurate data where naive algorithms produce inconclusive results.

Method

Subjects

Sixteen Carnegie Mellon students (8 women and 8 men) with normal, uncorrected vision participated in the experiment. All students were native English speakers. Two additional students were run but omitted from data analysis because their eye movements could not be reliably tracked.

Materials

The experiment materials comprised 48 medium-length (9- to 14-word) sentences identical to those in the original SRC experiment. For 12 of these sentences, a single true/false question was developed to test for basic understanding of the sentence. Eye-movement data were collected using an Iscan RK726/RK520 eye tracker with a Polhemus FASTRACK head tracker. The Iscan eye tracker uses the pupil-center and corneal-reflection points to estimate the angle of the left eye's gaze. The head tracker determines the position of the head and eye with respect to the visual field. The system collects eye and head information to produce a stream of on-screen point-of-regard locations. Data were sampled at a rate of 120 Hz. Subjects were seated approximately 30″ from the computer display screen. Characters in the stimuli were approximately 1/4″ in height (.48° of visual angle). The data were time-averaged by taking the median of the x and y coordinates with a 16-coordinate window; median averaging, unlike mean averaging used in the equation-solving study, helped to maintain the duration of fixations as accurately as possible.

Procedure

The experiment was run in one hour or less in a single session. Before starting, subjects were asked to read the sentences as quickly as possible and told that their understanding of the sentences would on occasion be tested. After eye-tracking calibration and a single test trial, subjects ran through the 48 trials with occasional breaks for calibration adjustment. Subjects triggered the trial by looking at a rectangle on the left side of the screen. When the sentence appeared, they read the sentence and signaled completion by looking at a rectangle on the right side of the screen. The experimenter manually terminated the trial when the estimated point-of-regard fell within this rectangle. For the 12 sentences with questions, subjects were then shown the true/false question and asked to type "T" for true and "F" for false. Eye movements were collected only during the reading phase for all trials.

Results without Tracing

We first analyze the New data without tracing to see how well they conform to the original SRC data. The data were analyzed with the naive method of identifying fixations and mapping fixations to the words on which they land; fixations landing on a space between words were mapped to the word on the right. Fixation identification was performed using I-HMM and the two-state velocity HMM shown in Figure 7.1; the velocity HMM differs from that in the equation-solving study due to the different time-averaging scheme used (see above). The HMM parameters were estimated as in the equation-solving study, yielding the transition values shown in the figure and the following other values: velocity mean and variance for the saccade state, 18.65 and 178.42; velocity mean and variance for the fixation state, 2.00 and 6.90; and prior state probabilities, .5.

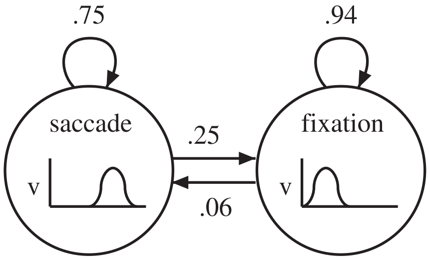


Figure 7.1: Velocity HMM for I-HMM fixation identification.

The following analysis centers on two aspects of the data including six different metrics: durations of gazes, first fixations, and single fixations; and probabilities of zero, one, or multiple fixations. These metrics are calculated with respect to five classes of word frequency: class 1: 1-10 occurrences per million, class 2: 11-100, class 3: 101-1000, class 4: 1001-10000, and class 5: 10001+. Like Reichle et al. (1998), we only analyze trials in which the eyes move strictly left-to-right (46% of all trials, compared to 36% in the SRC experiment); this restriction allows re-fixations but removes any effect of regressions, which are often caused by more complex syntactic or semantic processes not addressed by the E-Z Reader models. Also, data for the first and last words are omitted from analyses due to effects in starting and ending the sentence. Finally, because subjects performed very well on the true/false questions (93% correct), we assume that they were indeed reading and understanding the sentences and we ignore the true-false questions in future analyses.

The analysis of fixation durations and probabilities utilizes two metrics to indicate the correspondence between the New data and the SRC data. First, it utilizes correlations between the two data sets for the various measures with respect to word frequency class. The correlations help to show that the New data bring out similar effects of word frequency that were observed in the SRC data. Second, the analysis examines *spans* for each measure that indicate the difference between the lowest and highest values for the frequency classes. The spans complement the correlations in that they give some estimate of the size of the observed effects. One caveat should be noted with both of these measures: The two data sets were collected from different populations of subjects, and thus it not necessarily the case that they must show exactly the same effects. Nevertheless, both populations included college-age students at good universities, so we have ample reason to believe that at least the patterns of effects should generally remain the same. The size of the effects, as measured by the spans, may show less similarity, especially considering the difference in eye-tracking equipment used.

Fixation Durations

We generally expect that more frequent words—that is, words that appear more frequently in standard usage—require less processing time. One common measure of processing time emphasizes the duration of fixations on the words, with the tacit assumption that the eye generally processes the information over which it currently resides (Just & Carpenter, 1984). We utilize three variations of the fixation duration measure: gaze duration, first-fixation duration, and single-fixation duration. Gaze duration takes the durations of consecutive fixations on a single word and combines them into a single duration. First-fixation duration includes the duration of only the first fixation; in other words, it ignores subsequent fixations on the word. Single-fixation duration measures the duration of single fixations when there are no subsequent fixations on the same word.

Table 7.4 shows the average gaze, first-fixation, and single-fixation durations for both the SRC data and the New data. For gaze duration, the New data have a pattern very similar to the SRC data: class 1 (low-frequency) words have the longest duration, class 5 (high-frequency) words the shortest, and the others in between. Although the New data does not have as great a duration span between highest and lowest as the SRC data (44 ms vs. 79 ms), the two data sets have a very high correlation, R=.93, for gaze duration. For first-fixation duration, the New data show almost no pattern whatsoever. While the SRC durations decrease from low- to high-frequency words with a span of 40 ms, the New data show no real decrease and span only 8 ms. The lack of pattern in the New data comes through in the low correlation, R=.22. For single-fixation duration, the New data show moderate decrease from low- to high-frequency words with a span of 17 ms, still much lower than the span of 49 ms for the SRC data. The two sets show reasonably good correlation, R=.82.

Table 7.4: Gaze, first-fixation, and single-fixation durations by frequency class, in ms. SRC represents the original experiment, New represents the new experiment without tracing.

Frequency Class	Gaze		First Fixation		Single Fixation	
	SRC	New	SRC	New	SRC	New
1	293	276	248	228	265	247
2	272	253	234	220	249	235
3	256	243	228	223	243	236
4	234	242	223	227	235	240
5	214	232	208	224	216	230
Span	79	44	40	8	49	17
R	.93		.22		.82	

Fixation Probabilities

Just as we expect word frequency to influence fixation duration, we also expect it to influence how many times a word is fixated or whether it is fixated at all. We now analyze the data with respect to three related measures, namely the probability of no fixations (i.e., a skip), one fixation, or two fixations on words in different frequency classes. Table 7.5 shows the average probabilities for these measures for the SRC and New data sets. Although the spans for the New data are not quite as high as those for the SRC data, the correlations for all three measures are extremely high, $R \geq .99$. The New data even reproduce subtle effects in the SRC data, such as the fact that the probability of one fixation for class 2 is slightly higher than that for class 1 and 3.

Table 7.5: Probabilities of no fixation (skip), one fixation, and two fixations by frequency class. SRC represents the original experiment, New represents the new experiment without tracing.

Frequency Class	No Fixation (Skip)		One Fixation		Two Fixations	
	SRC	New	SRC	New	SRC	New
1	.10	.19	.68	.61	.20	.16
2	.13	.19	.70	.64	.16	.12
3	.22	.28	.68	.61	.10	.09
4	.55	.54	.44	.40	.02	.04
5	.67	.67	.32	.30	.01	.02
Span	.57	.48	.38	.34	.19	.14
R	.99		.99		.99	

Summary

Overall the New data look rather similar to the SRC data, providing good confirmation of the results of the SRC experiment and giving us confidence that the data nicely represent standard reading behavior. The New data reproduce the patterns of fixation probabilities almost exactly with very high correlations. However, with respect to fixation durations, the similarities between the New data and the SRC data are less clear: the gaze durations match very well, but single-fixation durations match only reasonably well, and first-fixation durations show almost no correspondence. The most likely reason for these discrepancies arises from the less accurate eye tracker used to collect the New data. Greater inaccuracies in the data may cause the naive algorithm to misattribute fixations to the wrong words—for instance, a slight calibration error may cause a long fixation on a low-frequency word to be observed on an adjacent high-frequency word, thus attenuating or eliminating potential frequency effects. As in the equation-solving study, we now develop tracing models for these data and subsequently show how these models help better understand and alleviate problems with misattributions.

Tracing Setup

Before tracing the reading data we must set up tracing with appropriate model grammars and parameter settings. The first step in our tracing setup entails estimating the E-Z Reader model parameters that maximize the models' fit to the data set. The second step involves running E-Z Reader model simulations and generating first- and second-order model grammars based on the simulations. The third step involves setting various tracing parameters for the Fixation-HMM and Point tracing methods. We discuss these steps in turn.

E-Z Reader Parameters

We utilize a procedure to estimate the E-Z Reader model parameters similar to that of Reichle et al. (1998). The estimated parameters f_b, f_m, and Δ were each varied across a reasonable range of values near those used by Reichle et al.; the E-Z Reader 5 parameters ε_1 and ε_2 were set to Reichle et al.'s best-fitting values. For each combination of parameter values, 300 model simulations were run for each of the 48 experiment sentences. These simulations were analyzed with respect to the six measures presented earlier: gaze, first-fixation, and single-fixation duration; and skip, one-fixation, and two-fixation probabilities. Finally, the root-mean-squared errors (RMSE) between the simulations and the New data were calculated and combined into a final RMSE as described in Reichle et al. (1998). The parameter value combination with the lowest RMSE was deemed the best combination for each model. The final parameter settings were as follows: E-Z Reader 3, $f_b = 230$, $f_m = 15$, $\Delta = .6$; E-Z Reader 5, $f_b = 150$, $f_m = 10$, $\Delta = .8$, $\varepsilon_1 = 1.25$, $\varepsilon_2 = 1.75$.

Model Grammars

As discussed in the introduction, enumerating every possible strategy (i.e., fixation sequence) for each sentence would create infeasibly large model grammars. Thus, it is necessary to build smaller model grammars that capture interesting aspects of the models' behavior while eliminating much of the unnecessary or less interesting information. A straightforward way to build such grammars is to translate the predictions of the models to first- and second-order grammars—that is, grammars that describe only the first- or second-order transitions between words. These grammars make the assumption that the next word to be fixated depends only on the current word (for a first-order grammar) or the current and previous word (for a second-order grammar). While the grammars clearly have less information than one that contains all enumerated strategies, they embody the most important sequential information in the models and provide an excellent starting point for tracing the reading data.

The first- and second-order grammars were generated from the E-Z Reader models as follows. First, for each sentence, the models were run for 300 simulations and the resulting fixation sequences were collapsed into first- and second-order transitions; simulations with regressions to previous words were discarded. Because the E-Z Reader models start by fixating the first word, the model grammars assume that the first fixation always occurs on this word. The transitions for the first-order grammar comprised first-order transitions from each word to itself and every word to its right. The transitions for the second-order grammar comprised first-order transitions from the first word to all other words and second-order transitions from a pair of words to the second word and words to its right. Probabilities for the transitions were derived directly from the model simulations; in other words, each transition probability represented the number of times the transition occurred divided by the number of opportunities in which the transition could occur. This entire process results in the creation of a first- and second-order grammar for each of the 48 sentences.

To better illustrate this building process, let us briefly look at (contrived) first- and second-order grammars for the sample sentence "See Jane run." (The sentence is rather short to keep the grammars small.) Table 7.6 shows a first-order grammar for this sentence; subgoals (non-terminals) are shown in italics and fixations (terminals) in bold. Each subgoal name represents the previously-fixated word. The first rule fixates the word **See** and sets the subgoal *see* to indicate that this was the last word fixated. The rules for the *see* subgoal allow the model to fixate any of the words next, or the special **end** target rectangle that terminals the trial. The probabilities of these rules are set such that the model most likely fixates the next word, **Jane**. After one of the *see* rules fires, the next word is fixated and a new subgoal is set that represents the fact that this word was fixated. Note that the rules only allow transitions back to the word or to later words.

Table 7.6: Sample first-order grammar for "See Jane run."

Rule			Probability
start	-->	**See** *see*	1.00
see	-->	**See** *see*	.10
see	-->	**Jane** *jane*	.50
see	-->	**run** *run*	.30
see	-->	**end**	.10
jane	-->	**Jane** *jane*	.10
jane	-->	**run** *run*	.70
jane	-->	**end**	.20
run	-->	**run** *run*	.10
run	-->	**end**	.90

Table 7.7 shows a sample second-order grammar for the same sentence. Again the first rule fixates **See** and sets the subgoal *see* to denote this history. The next four rules for the subgoal *see* are first-order rules that fixate another word or the **end** target. When one of the *see* rules fires, the model fixates the word and sets a second-order subgoal; this subgoal represents a history of two words, namely the previous word **See** and the currently-fixated word. The remaining rules, all second-order, continue firing until the **end** target is fixated; all second-order subgoal names denote the previous two words fixated. Rules for the final four subgoals are not

completely specified but follow the pattern of the other second-order rules. This second-order grammar, unlike the first-order grammar, allows transition probabilities to depend on the last word before the current word and thus allows us to compare whether such information benefits tracing in any way.

Table 7.7: Sample second-order grammar for "See Jane run."

Rule			Probability
start	-->	**See** *see*	1.00
see	-->	**See** *see-see*	.10
see	-->	**Jane** *see-jane*	.50
see	-->	**run** *see-run*	.30
see	-->	**end**	.10
see-see	-->	**See** *see-see*	.10
see-see	-->	**Jane** *see-jane*	.50
see-see	-->	**run** *see-run*	.30
see-see	-->	**end**	.10
see-jane	-->	**Jane** *jane-jane*	.10
see-jane	-->	**run** *jane-run*	.70
see-jane	-->	**end**	.20
see-run	-->	< 2 rules >	...
jane-jane	-->	< 3 rules >	...
jane-run	-->	< 2 rules >	...
run-run	-->	< 2 rules >	...

Tracing Parameters

The final step of the tracing setup is setting the specific parameters for the tracing methods. As we use only the HMM-based methods Fixation-HMM and Point, we have only to specify the form of the fixation HMMs for these methods, as shown in Figure 7.1. Because the reading study requires analysis at the level of fixations, it need not utilize the same fixation HMMs as in the equation-solving study, since those HMMs represented gazes rather than fixations. Figure 7.1(a) shows the fixation HMM used for Fixation-HMM, and Figure 7.1(b) shows that used for Point.

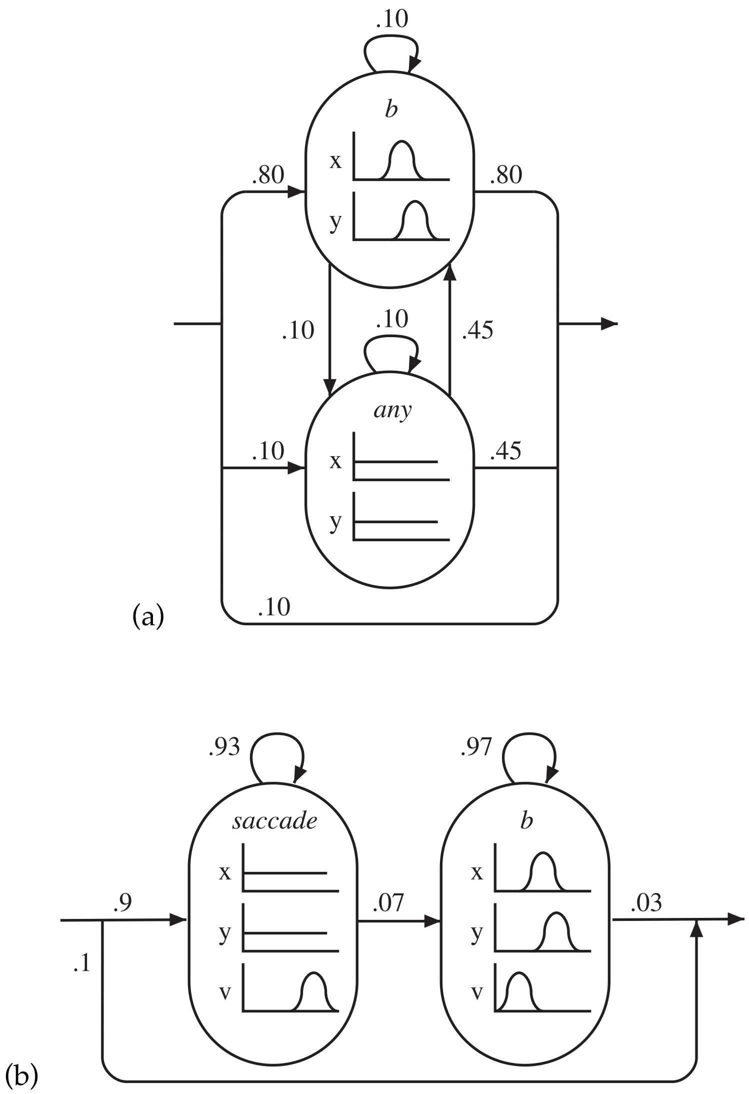


Figure 7.2: Fixation HMMs used with (a) Fixation-HMM and (b) Point.

Model Comparison

In their development of the E-Z Reader models, Reichle et al. (1998) found that E-Z Readers 3 and 5 fit the Schilling et al. (1998) data equally well with respect to the aggregate fixation duration and probability measures discussed earlier. However, Reichle et al. had a strong qualitative reason to believe that E-Z Reader 5 was the "state-of-the-art" model: it accounted for the fact that words farther from the fovea (i.e., farther from the currently-fixated word) are more difficult to encode and process. In essence, this difference was primarily one of sequential information, in that E-Z Reader 5 exhibits more difficulty encoding parafoveal words and thus produces shorter saccades and more fixations. Although Reichle et al.'s brief discussion of sequential effects suggests that E-Z Reader 5 captured sequential behavior better than E-Z Reader 3, the comparison was generally "encouraging" but not conclusive.

We now investigate whether tracing can help compare the two models with respect to sequential information. By tracing the data with the models' first- and second-order grammars, we can derive mismatch scores that represent how well each model fits the data. (Recall that the mismatch scores for fixation and point tracing are based on the probability of the eye-movement protocols given the tracer model HMM; see Chapter 4 for details.) Comparison of the mismatch scores for similar grammars derived from E-Z Readers 3 and 5 provide a direct quantitative test of which model better captures sequential aspects of the data.

Table 7.8 shows the average mismatch scores for the first- and second-order grammars for Fixation-HMM and Point. In all cases, E-Z Reader 5 exhibits a lower mismatch score and thus provides a better fit to the data. Repeated-measure ANOVAs show that the differences are very significant for first-order Fixation-HMM, second-order Fixation-HMM, and second-order Point, p<.0001, and moderately significant for first-order Point, p<.1. The differences are quite small for Point, due primarily to the fact that the second-order predictions of the models are so similar; nevertheless, the small differences hold for almost all individual subjects, producing a significant effect. Note that scores are not directly comparable between methods, because of the different fixation HMMs, or between grammars, because of lower transition probabilities in the second-order grammar. This quantitative comparison of model fit thus supports and confirms Reichle et al.'s qualitative argument.

Table 7.8: Mismatch scores for the first- and second-order grammars using Fixation-HMM and Point. Note that scores are not directly comparable between methods or grammars.

First-order Grammar				Second-order Grammar			
Fixation-HMM		Point		Fixation-HMM		Point	
EZR 3	EZR 5	EZR 3	EZR 5	EZR 3	EZR 5	EZR 3	EZR 5
11.824	11.780	13.961	13.959	11.913	11.851	13.962	13.959

Results with Tracing

This section revisits the earlier results derived without tracing to see whether and how tracing clears up understanding of the New data. Again we analyze the data with respect to gaze, first-fixation, and single-fixation durations as well as skip, one-fixation, and two-fixation probabilities. Because the above comparison indicates that E-Z Reader 5 better captures subject behavior, the following analyses use the interpretations of its first- and second-order grammars;

the interpretations of E-Z Reader 3 are highly similar and contribute little to this particular exposition.

Fixation Durations

Without tracing, the fixation duration results showed that gaze durations were very similar between the SRC and New experiments, single-fixation durations were somewhat similar, and first-fixation durations were dissimilar. Tracing helps to clean up the New data such that it reveals similar patterns to the SRC data for all measures. Table 7.9 shows the gaze duration results for the SRC data, the New data without tracing, and the New data traced in four ways: with a first- or second-order grammar, and with Fixation-HMM or Point. Tracing generally provides little to no improvement for this measure. The correlations remain high for the four methods, $R \geq .90$, and the span (the size of the overall effect) remains similar to that without tracing. The magnitudes of the gaze durations traced with Point are similar to those without tracing, but those traced with Fixation-HMM are generally slightly lower (by approximately 20 ms). In addition, the first-order results are extremely similar to the second-order results. We address these issues in the upcoming discussion section.

Table 7.9: Gaze durations for the SRC and New data.

Frequency Class	SRC	New				
		No Model	First-order		Second-order	
			F-HMM	Point	F-HMM	Point
1	293	276	251	272	248	272
2	272	253	232	253	229	252
3	256	243	230	248	229	247
4	234	242	225	248	229	249
5	214	232	208	223	207	221
Span	79	44	43	49	41	51
R	-	.93	.96	.93	.90	.92

Although tracing helps little with gaze durations, it helps significantly with first-fixation durations and somewhat for single-fixation durations. Tables 7.10 and 7.11 show the results for first-fixation and single-fixation durations, respectively. For first-fixation durations, tracing helps bring the correlation between the SRC and New results from .22 without tracing to .81-.87 with tracing. In addition, tracing improves the span results from 8 ms without tracing to 21-25 ms with tracing. For single-fixation durations, tracing improves the correlations from .82 to the range .87-.95 and approximately doubles the span results. Both measures, like the SRC results, show higher durations for class 1 (low-frequency) words, lower durations for class 5 (high-frequency) words, and in-between durations for classes 2-4. Thus, not only does tracing improve the similarity of the pattern of results, but also the magnitude of the effects as measured by span.

Table 7.10: First-fixation durations for the SRC and New data.

Frequency Class	SRC	New				
		No Model	First-order		Second-order	
			F-HMM	Point	F-HMM	Point
1	248	228	226	247	226	246
2	234	220	212	241	213	239
3	228	223	218	242	219	241
4	223	227	216	244	218	244
5	208	224	205	222	202	221
Span	40	8	21	25	24	25
R	-	.22	.87	.84	.87	.81

Table 7.11: Single-fixation durations for the SRC and New data.

Frequency Class	SRC	New				
		No Model	First-order		Second-order	
			F-HMM	Point	F-HMM	Point
1	265	247	240	258	239	257
2	249	235	222	244	221	243
3	243	236	227	245	227	244
4	235	240	223	246	229	247
5	216	230	206	223	205	221
Span	49	17	34	35	34	36
R	-	.82	.95	.94	.87	.91

Fixation Probabilities

The results for fixation probabilities without tracing showed excellent correlations between the SRC and New data. We hope and expect that tracing maintains these correlations. Tables 7.12, 7.13, and 7.14 show the results with and without tracing for probabilities of skips, one fixation, and two fixations. The results for all measures show that correlations remain high, *R*≥.93, for all tracing methods and grammars. For the skip and one-fixation probabilities, the spans become slightly lower when traced with Point and slightly higher when traced with Fixation-HMM. For the two-fixation probabilities, the spans with tracing are approximately half the span without tracing; in conjunction with the higher one-fixation probabilities, this seems to show that the tracing methods have a tendency to split multiple fixations on the same word into fixations on different words. Overall, however, tracing maintains the high correlations and similar spans between the SRC and New data.

Table 7.12: Skip probabilities for the SRC and New data.

Frequency Class	SRC	New				
		No Model	First-order		Second-order	
			F-HMM	Point	F-HMM	Point
1	.10	.19	.13	.21	.12	.21
2	.13	.19	.17	.23	.15	.23
3	.22	.28	.27	.29	.26	.28
4	.55	.54	.46	.42	.46	.42
5	.67	.67	.71	.58	.71	.57
Span	.57	.48	.58	.37	.59	.36
R	-	.99	.98	.98	.98	.98

Table 7.13: One-fixation probabilities for the SRC and New data.

Frequency Class	SRC	New				
		No Model	First-order		Second-order	
			F-HMM	Point	F-HMM	Point
1	.68	.61	.76	.70	.77	.70
2	.70	.64	.73	.72	.76	.72
3	.68	.61	.66	.68	.67	.70
4	.44	.40	.50	.57	.50	.56
5	.32	.30	.28	.41	.28	.42
Span	.38	.34	.48	.31	.49	.30
R	-	.99	.97	.98	.97	.99

Table 7.14: Two-fixation probabilities for the SRC and New data.

Frequency Class	SRC	New				
		No Model	First-order		Second-order	
			F-HMM	Point	F-HMM	Point
1	.20	.16	.08	.07	.09	.08
2	.16	.12	.08	.04	.06	.05
3	.10	.09	.05	.03	.05	.02
4	.02	.04	.03	.01	.03	.02
5	.01	.02	.01	.00	.01	.00
Span	.19	.14	.07	.07	.08	.08
R	-	.99	.97	.97	.96	.93

Summary

By using sequential information to sort out temporal information, tracing facilitates the analysis of the New data with respect to fixation durations and probabilities. The benefits of tracing are clearest for two measures, first-fixation and single-fixation duration, for which the traced interpretations provide better correspondence with the SRC data and strengthen effects observed in the SRC data. For the other measures, tracing seems to provide no real benefit but maintains at least as good results as the analysis without tracing.

Discussion

We have explored two ways in which tracing facilitates the study and modeling of eye movements in reading. First, tracing allows for quantitative comparison of reading models with respect to their sequential behavior. The comparison of E-Z Readers 3 and 5 was very successful in that all methods of tracing pointed to a clear winner, E-Z Reader 5, and this winner had been predicted qualitatively to more closely capture reading behavior. Second, tracing helps to sort out temporal information by guiding the interpretation of reading eye movements through a model's sequential information. The use of the E-Z Reader models to clean up the first-fixation and single-fixation results shows that the tracing methods can assist in analysis and understanding of reading data even in the face of eye-tracker noise and individual variability.

Evaluating the Sequential Behavior of Reading Models

One issue that arises in the study of eye movements in reading is the difficulty of evaluating reading models with respect to their sequential behavior. This study has emphasized one way to address this issue, namely the direct evaluation of models by tracing and deriving a goodness-of-fit metric. However, the study indirectly suggests another powerful way to attack this problem: examination of the models' sequential characteristics as described by their first-order or second-order transitions. In developing the E-Z Reader model grammars, we ran a number of stochastic simulations of the models and compiled the simulations to derive grammars from first- and second-order transitions. By examining the information in these transitions, we can better understand the sequential behavior of the models and compare these behaviors with other competing models. Though not common, this type of transition analysis (or variants thereof) have been employed by several researchers; for instance, Legge, Klitz, & Tjan (1997) examined the saccade lengths of their Mr. Chips model in units of letters, in essence collapsing first-order transitions between letters into a single aggregate measure.

To illustrate this idea, let us briefly perform a first-order transition analysis of the New data (without tracing) and E-Z Readers 3 and 5, as shown in Table 7.15. The table describes the probability of making saccades (i.e., transitions) of various lengths, measured in number of words; the results include only left-to-right scans and ignore fixations on the first and last words of a sentence. The New data transitions are heavily weighted near the current word, with a high probability of re-fixating the current word (length 0) or fixating the next or second-next word (lengths 1-2). The E-Z Reader models exhibit less of a tendency to re-fixate and more of a tendency to fixate the next word. The probabilities for both the data and the models decrease dramatically for lengths 3 and greater. Nevertheless, the models' prediction correlate well with the data, with E-Z Reader 5 producing a better correlation, *R*=.89, and error, *RSME*=.082, than E-Z Reader 3, *R*=.83, *RSME*=.105. This brief analysis thus supports earlier ones in determining that E-Z Reader 5 provides the better model fit in terms of sequential aspects of the data.

Table 7.15: One-transition probabilities for the New data (untraced) and E-Z Readers 3 and 5. Columns *R* and *RSME* represent correlations and root-mean-squared errors between model and data.

Source	Saccade Length						*R*	*RSME*
	0	1	2	3	4	5		
New data	.21	.39	.31	.08	.01	.00		
EZR 3	.12	.63	.16	.06	.02	.01	.83	.105
EZR 5	.11	.58	.22	.08	.01	.00	.89	.082

While transition analysis helps evaluate and compare models of reading, we should note two important differences between transition analysis and tracing. First, tracing forms new interpretations of protocols based on sequential information and thus may be used to clean up other aspects of behavior, such as temporal information. Second, whereas transition analysis examines only the probabilities of transitions, tracing utilizes the probabilities of both transitions and fixation locations. Tracing is thus able to make more fine-grained evaluations based on the interaction of transition and fixation-location probabilities. For instance, consider a fixation very close to the border between two targets and consider how noise in the fixation's sampled location affects the two types of analyses. Tracing is very robust in this case: a slight movement of the fixation creates a small change in the location probabilities (i.e., the x and y probabilities for the fixation state), and thus only a small change in the overall evaluation. Transition analysis, in contrast, must assign the fixation to one target or the other; a slight change within a target area has no affect on overall evaluation, whereas a slight change that moves the fixation from one target area to the other can have a dramatic change in evaluation. Overall, transition analysis seems more suitable for coarser evaluations and tracing may be preferable when more robust analysis is required.

Comparing First-Order and Second-Order Transition Information

Another issue in our analysis of the reading data arises in whether the second-order model grammars offer any more power or flexibility than the first-order grammars. Both grammars provided equivalent results in almost every aspect of the analyses to this point: both classified E-Z Reader 5 as the better model, and both improved the temporal aspects of the New data in very similar ways. Thus, we turn to another type of analysis to attempt to discriminate between the two grammars by comparing the "observed" and "computed" second-order transition matrices. For these purposes we define a second-order transition matrix of values $< i , j >$ to encode the probabilities of making a saccade of distance i followed by a saccade of distance j. The observed matrix simply represents these probabilities as observed in the data source (empirical data or model simulations). The computed matrix represents these probabilities as derived from the first-order transition probabilities: each entry $< i , j >$ is computed as the product of probabilities of length-i and length-j saccades (in Table 7.15). By comparing the similarity of the observed and computed matrices, we can get some idea of how much extra predictive power can be derived from the second-order transition matrix over the first-order transitions.

Table 7.16 shows the correlations and errors between the observed and computed second-order transition matrices for the New data and the E-Z Readers 3 and 5, where the matrices represent saccades of length 0 to 3. The table shows extremely high correlations, $R \geq .98$, and low errors, $RSME < .05$, between the observed and computed matrices. Thus, for the New data set and the E-Z Reader models, the first-order transitions capture almost all the information present in the second-order transitions. Of course, both these data and models represent only left-to-right reading scans, so at best we can claim that second-order transitions seem to provide little benefit over first-order transitions for these scans. It remains an open and interesting question whether this result would hold for non-left-to-right scans.

Table 7.16. Correlations and errors between observed and computed second-order transition matrices for the New data (untraced) and E-Z Readers 3 and 5.

Source	*R*	*RSME*
New data	.98	.035
EZR 3	.99	.047
EZR 5	.99	.048

Comparing Fixation Tracing and Point Tracing

The model comparison and the results with tracing in the previous two sections showed that the choice of tracing methods can affect downstream analysis. However, both fixation tracing (embodied in Fixation-HMM) and point tracing (embodied in Point) showed strengths and weaknesses, making it difficult to name one as the better method for the reading domain. In the model comparison, fixation tracing provided very significant evaluation differences for both the first-order and second-order grammars, while point tracing provided only mildly significant differences for the first-order grammar. However, in the results with tracing, point tracing had a lower average error for fixation durations, *RMSE*=13.1, and probabilities, *RMSE*=1.3, than fixation tracing, *RMSE*=24.5, *RMSE*=3.3, respectively. Thus, fixation tracing seemed to perform better for the model comparisons and point tracing for the temporal results. We explore the comparison of fixation and point tracing further in the final chapter.

Chapter 8

Eye Typing

Introduction

In the previous two chapters, we examined eye movements as a tool for understanding and modeling cognitive processes. In this chapter, we explore a seemingly orthogonal use of eye movements: eye-based user interfaces in which users communicate with a computer by means of their eye movements. Researchers have recently developed such interfaces for a wide variety of tasks, including word processing (e.g., Frey, White, & Hutchinson, 1990; Gips, 1998; Hutchinson, White, Martin, Reichert, & Frey, 1989; Stampe & Reingold, 1995), musical composition (Gips, 1998), control of household lights and appliances (e.g., Cleveland, 1997), and control of robotic wheelchairs (e.g., Yanco, 1998). While eye-based interfaces may appear to have little in common with the study and modeling of cognition, the two in fact share an important commonality—the need to interpret human actions by mapping actions to the thoughts that produced them. However, eye-based interfaces must interpret actions on-line and in real-time, making them an excellent application domain in which to test both the speed and accuracy of tracing.

This chapter demonstrates how tracing assists in the interpretation of user input in eye-based interfaces and thus facilitates the design, implementation, and usability of these interfaces. The chapter focuses on a study of a particular eye-based interface, an *eye-typing* interface, in which users type characters by looking at characters on an on-screen keypad. Previous eye-typing systems (e.g., Frey, White, & Hutchinson, 1990; Gips, 1998; Hutchinson et al., 1989; Stampe & Reingold, 1995) have enjoyed a great deal of success enabling physically-disabled users to type using their eye movements alone. However, they incorporate restrictions to facilitate the interpretation of user input, sacrificing speed and usability in the process. This study examines a prototype eye-typing interface that minimizes these restrictions and, as expected, makes the interpretation of user input extremely difficult. The study uses this interface to explore how difficult interpretation becomes when restrictions are removed and how the tracing methods can assist in interpreting this input. In addition, the study investigates the speed-accuracy tradeoffs of the various tracing methods and discusses their applicability to real-time eye-based interfaces.

While this chapter discusses tracing only with respect to eye-based interfaces, the tracing of eye movements can clearly contribute to the development of other, more typical human-computer interfaces. One application involves using tracing, or some form thereof, to interpret user protocols and identify task behaviors and strategies (e.g., Aaltonen, Hyrskykari, & Räihä, 1998; Byrne, Anderson, Douglass, & Matessa, 1999). This application is very similar to the studies of equation solving and reading, in that tracing assists in analysis of empirical data to elicit and understand behavior at a fine-grain level. Another application entails the incorporation of tracing into intelligent interfaces that interpret user knowledge and adapt accordingly. A good example of such an interface is an intelligent tutoring system (e.g., Anderson, Corbett, Koedinger, & Pelletier, 1995; Frederiksen & White, 1990) that interprets student responses to infer the student's current state of knowledge. Again, the tracing of eye movements can help produce more accurate estimations of knowledge state and more appropriate responses to user actions.

Eye-Typing Study

The eye-typing study focuses on an eye-typing interface that allows users to type words using their eye movements alone. Past work on eye-typing systems, such as those cited above, have demonstrated the great potential for such systems to facilitate hands-free typing. However, they have also manifested three severe problems with respect to interpreting eye movements:

- Off-center fixations: Observed fixations as recorded by eye trackers are often not centered over visual targets because of foveal and parafoveal encoding, eye-tracker noise, and eye-tracker calibration error or degradation. This problem requires that interfaces recognize off-center fixations and map them to the correct intended targets.

- Incidental fixations: As users fixate various targets to actuate commands, they also produce incidental fixations (e.g., during search) that are not intended to actuate commands. This so-called "Midas touch" problem (Jacob, 1995) requires that interfaces distinguish intended command fixations from incidental non-command fixations.

- Skipped fixations: When attempting to look at various targets, users can move their attention between nearby targets without actually moving their eyes, apparently skipping fixations on targets. This separation of attention and eye-gaze location (Pashler, 1998) requires that interfaces predict when users attend to multiple targets during the same fixation.

These problems are especially critical in eye-based interfaces because the interfaces provide feedback based on user eye movements, making accurate interpretation essential for a user-friendly system.

To cope with these problems, existing systems incorporate two major restrictions. First, the systems typically require large distances—approximately 4° of visual angle—between visual targets. The coarse spacing alleviates the problems of off-center fixations but limits the amount of information presented on the screen. Second, the systems require users to fixate visual targets for long durations—750 ms to a few seconds—to trigger an action. These long durations alleviate incidental and skipped fixations but result in extremely slow input times. Thus, the restrictions of large distances and long durations seriously limit the usability of these interfaces.

This study aims to determine the implications of eliminating these restrictions to allow for more freedom in the design and use of eye-based interfaces. The study has two primary goals. First, the study evaluates how eliminating the restrictions hinders interpretation of user eye movements. Because existing systems impose these restrictions to facilitate interpretation, it is important to see how difficult interpretations become when the restrictions are removed. Second, the study tests how well the proposed tracing methods can interpret protocols in the context of a less restrictive interface. The performance of tracing is compared with that of the standard naive algorithm in existing systems to measure any potential improvement offered by tracing. In addition, the study has several secondary goals, including examination of tracing's scalability to large vocabularies, analysis of user improvement with practice, and comparison between eye typing and hand typing.

The eye-typing system used in the study was designed primarily for data collection of eye movements during an eye-typing task. In the task, users encounter a screen containing the word to be typed at the top, the typing keypad in the middle, and the output box at the bottom, as shown in Figure 8.1. Each trial begins with the user reading the word to be typed. To type the word, the user fixates the letters of the word in sequence on the on-screen keypad. For repeated letters, the user first fixates the letter and then fixates the "Repeat" key at the bottom-right of the keypad. After fixating the final letter, the user looks down into the output box displayed beneath the keypad to terminate the trial.

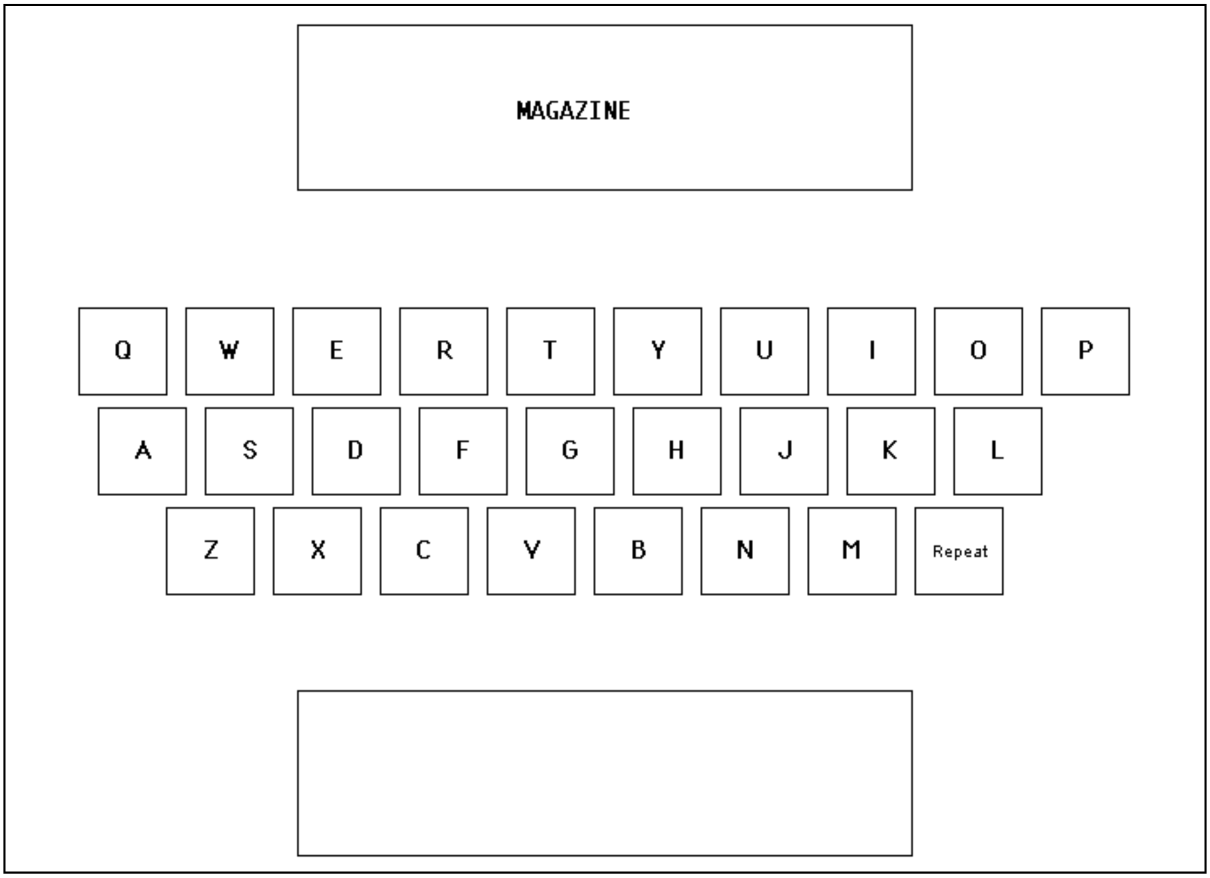


Figure 8.1: Eye-typing interface screen.

The eye-typing system interface used in the study eliminates both the spacing and time restrictions mentioned earlier. First, the interface spaces keypad letters by approximately 1.5° visual angle to reflect the size of the fovea (Fuchs, 1971); this tight spacing provides a rigorous test for the tracing algorithms in handling off-center fixations. Of course, we would expect better end results with a looser spacing, but the tight spacing better tests the limits of the algorithms. Second, the interface allows the user to fixate visual targets for an arbitrary duration—that is, the user can move her/his eyes as quickly as desired. This allowance also provides a rigorous test for the tracing algorithms: The system can no longer rely on duration to discriminate incidental from command fixations, but instead must employ the user model to make this discrimination.

Method

Subjects

Seven Carnegie Mellon students (4 women and 3 men) with normal, uncorrected vision participated in the experiment. Two additional students were run but omitted from data analysis because their eye movements could not be reliably tracked.

Materials

The experiment materials included 12 words to be typed: *ANIMALS*, *FOREST*, *KITCHEN*, *LETTER*, *MACHINE*, *MAGAZINE*, *MOTHER*, *PLANTS*, *SENATOR*, *SQUARE*, *STREET*, and *TEACHER*. The words were eye-typed on a keypad identical to that shown in Figure 8.1; the screen also included the word to be typed above the keypad and an output box below the keypad. Eye-movement data were collected using an Iscan RK726/RK520 eye tracker with a remote camera and a chin rest to stabilize the head. Data were sampled at a rate of 60 Hz and time-averaged with a window of 4 data points. Subjects were seated approximately 30″ from the computer display screen. Characters in the keypad were approximately 1/4″ in height (0.48° of visual angle) and separated by approximately 7/8″ (1.66° of visual angle).

Procedure

The experiment was run in one hour or less in a single session. Before starting, subjects were instructed to type the given words as quickly and accurately as possible. After eye-tracking calibration and a single test trial, subjects eye-typed 12 words four times (a total of 48 trials) and hand-typed the same 12 words once; the order of eye and hand typing was counterbalanced. For eye-typing trials, subjects looked at the given word above the keypad, typed the word, and looked at the output box below the keypad, at which time the experimenter manually terminated the trial. For hand-typing trials, subjects typed the word on the keyboard (with the alphabetic and delete keys enabled) and hit return to end the trial. Eye movements were collected only during the eye-typing trials.

Results without Tracing

As in the equation-solving and reading studies, we first examine the results of the experiment without tracing. This analysis looks at two aspects of these results. First, it presents several illustrative protocols that demonstrate the difficulty of interpreting eye movements when interface restrictions are eliminated. Second, it provides a brief look at accuracy when interpreting protocols using the standard naive algorithm.

Illustrative Protocols

Figure 8.2 shows three sample protocols for subjects typing the words *SQUARE*, *TEACHER*, and *MOTHER*. These protocols illustrate some of the major problems of eye-typing protocols discussed earlier. Figure 8.2(a) includes two incidental fixations (4 and 6), both of which represent a saccadic undershoot on the way to the desired target. The protocol also includes several off-center fixations, the worst of which (3) lies closest to a presumably unintended target (*A* instead of *Q*). Figures 8.2(b) and 8.2(c) further illustrate these problems and the extent to which they may be present in the protocols. In Figure 8.2(b), the subject's fixations on *C* (4) and *H* (5) are far off-center, and her encoding of *E* and *A* occur during a single fixation (3). In Figure 8.3(c), the subject exhibits a large number of incidental fixations, totaling 16 fixations for a 6-letter word. Naive interpretation of these protocols (mapping fixations to their nearest targets) generates the typed words *SAYUEARE*, *TWFGRE*, and *TBNMFDSUIOYTHER*. Such protocols obviously create great difficulty for naive interpretations.

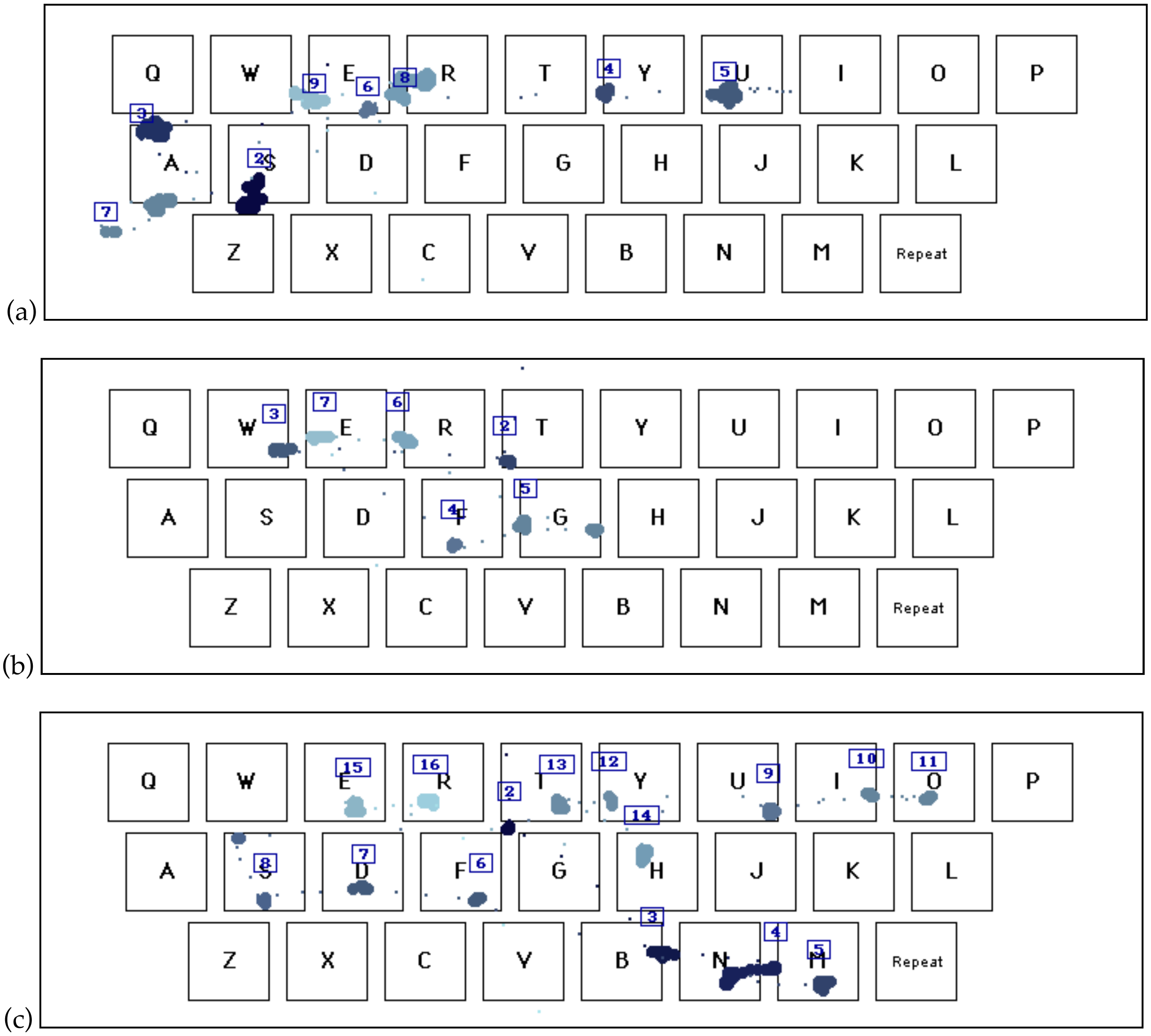


Figure 8.2: Sample eye-typing protocols for the words (a) *SQUARE*, (b) *TEACHER*, and (c) *MOTHER*.

Interpretation Accuracy

To quantify the difficulty of interpreting the protocols, we can look at the accuracy of naive algorithms as how often their interpretations match the word given to be typed in a particular trial. Fixations were identified using the four fixation-identification methods in

Chapter 3 and mapped to the closest target area. With these interpretations, all four methods averaged no more than 1% accuracy on the entire protocol data set. Clearly, naive interpretation of these unrestricted eye movements is not feasible; more sophisticated methods such as tracing are required to correct for off-center, incidental, and skipped fixations.

Tracing Setup

Tracing allows us to interpret the eye-typing protocols by mapping each protocol to one of a set of possibly-intended words. While subjects typed only 12 different words, we can force tracing to classify each protocol as one of any number of words. Our analysis examines vocabularies of 12, 100, and 1000 words to explore how tracing accuracy and speed degrade for larger vocabularies; of course, all vocabularies include the 12 words tested in data collection. We now specify the model grammars for these vocabularies and the parameters used in tracing.

Model Grammars

The analysis of the eye-typing protocols utilizes four models of user behavior for each vocabulary. The four models are as follows:

- Full: represents each word separately
- Second-order: represents transitions from letter pair to letter
- First-order: represents transitions from letter to letter
- Simple: maps each fixation to the nearest letter

These four models provide varying amounts of sequential information: the full model gives a full description of possible action sequences, the simple model gives no information about action sequences, and the first-order and second-order models fall in between these two extremes. As we will see, models with more sequential information produce more accurate interpretations but require more time to generate interpretations.

To better illustrate these four types of models, let us examine each type for a small vocabulary of three words: *RAT*, *TRAP*, and *PART*. The full model contains a single rule for each word in the vocabulary, as shown in Table 8.1. The model first fixates **word** (the given word to be typed) and moves to one of the word subgoals with equal probability. The model then types the word's letters and finally fixates **out** (the output box), signaling completion of a trial. In actual implementation, the grammar can be compacted so that words with the same prefixes use the same subgoals; this modification results in smaller HMMs, especially for large vocabularies.

Table 8.1: Full grammar for {*RAT*, *TRAP*, *PART*}.

Rule			Probability
start	→	**word** *rat*	.33
start	→	**word** *trap*	.33
start	→	**word** *part*	.33
rat	→	**R A T** *end*	1.00
trap	→	**T R A P** *end*	1.00
part	→	**P A R T** *end*	1.00
end	→	**out**	1.00

The second-order model represents second-order transitions from letter pairs to letters, as shown in Table 8.2. After fixating the target **word**, the model moves to the subgoal for a letter with the probability of that letter starting a word. The model then moves to second-order rules that, for each letter pair, fixate the second letter and transition to a new letter pair. This type of encoding can contain very useful sequential information, such as the fact that consonant clusters like *TR* almost always precede vowels. However, the second-order model is not as fully specified as the full model because it allows for both English and non-English words not in the vocabulary—for instance, *RAP* and *TRAT*.

Table 8.2: second-order grammar for {*RAT*, *TRAP*, *PART*}.

Rule			Probability
start	→	**word** *p*	.33
start	→	**word** *r*	.33
start	→	**word** *t*	.33
p	→	**P** *pa*	1.00
r	→	**R** *ra*	1.00
t	→	**T** *tr*	1.00
ap	→	**P** *end*	1.00
ar	→	**R** *rt*	1.00
at	→	**T** *end*	1.00
pa	→	**A** *ar*	1.00
ra	→	**A** *ap*	.50
ra	→	**A** *at*	.50
rt	→	**T** *end*	1.00
tr	→	**R** *ra*	1.00
end	→	**out**	1.00

The first-order model represents first-order transitions from one letter to another, as shown in Table 8.3. Like the second-order model, the first-order model starts by moving to a letter subgoal with the appropriate probability. Then, the model simply transitions from letter to letter with the probability observed in the vocabulary. Again, this type of model can encode useful information about the vocabulary, such as the fact the Q almost always precedes U. However, this model contains even less information than the second-order model; for instance, it can produce the words *RAP* and *TRAT*, like the second-order model, but can also produce *PAT* and *TRT*, unlike the second-order model.

Table 8.3. first-order grammar for {*RAT*, *TRAP*, *PART*}.

Rule			Probability
start	→	**word** *p*	.33
start	→	**word** *r*	.33
start	→	**word** *t*	.33
a	→	**A** *p*	.33
a	→	**A** *r*	.33
a	→	**A** *t*	.33
p	→	**P** *a*	.50
p	→	**P** *end*	.50
r	→	**R** *a*	.67
r	→	**R** *t*	.33
t	→	**T** *r*	.33
t	→	**T** *end*	.67
end	→	**out**	1.00

The simple model maps each fixation to its nearest letter target. This model, or a similar form thereof, represents the naive algorithm typically used in existing eye-based interfaces. The simple model can be represented as a first-order grammar with uniform transition probabilities for every letter. However, there is a simpler and more efficient implementation: after finding fixation centroids, simply compute the nearest target to each centroid and labels the fixation with these nearest targets. The simple model contains essentially no sequential information and is thus the antithesis of the full model.

Tracing Methods

Using the four model grammars for each vocabulary, we can trace the eye-typing protocols using the tracing methods built into EyeTracer (see Chapter 5). There are two issues that arise in tracing with the various tracing methods. First, while fixation and point tracing can utilize all four models, target tracing can only utilize the full and simple models because it requires that the grammar be expanded into all possible sequences. Second, of the four models, only the full model is guaranteed to produce a model trace that actually represents a word in the vocabulary. For the other models, in addition to tracing in the usual manner, we also map the resulting trace to a vocabulary word by means of sequence matching. This method of tracing is, in some sense, a combination of the HMM-based and the sequence-matching tracing methods. This issue is discussed further in the final chapter.

Tracing Parameters

The final step in the tracing setup is the setting of the various tracing method parameters. Target-HMM used the velocity fixation-identification HMM shown in Figure 8.3, with a saccade mean and variance of 26.53 and 279.52, and a fixation mean and variance of 1.95 and 2.32, respectively. The other target tracing methods used the default parameters as set in the equation-solving study: Target-VT, velocity threshold = 5; Target-DT, dispersion threshold = 10, duration threshold = 50; Target-TA, duration threshold = 50. Fixation-HMM and Point used the fixation HMMs shown in Figure 8.4(a) and 8.4(b), respectively. The fixation HMMs differ from those in equation solving and reading to produce faster interpretation times as is critical for this study.

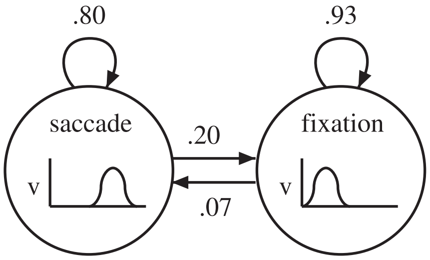


Figure 8.3: Velocity HMM for HMM fixation identification.

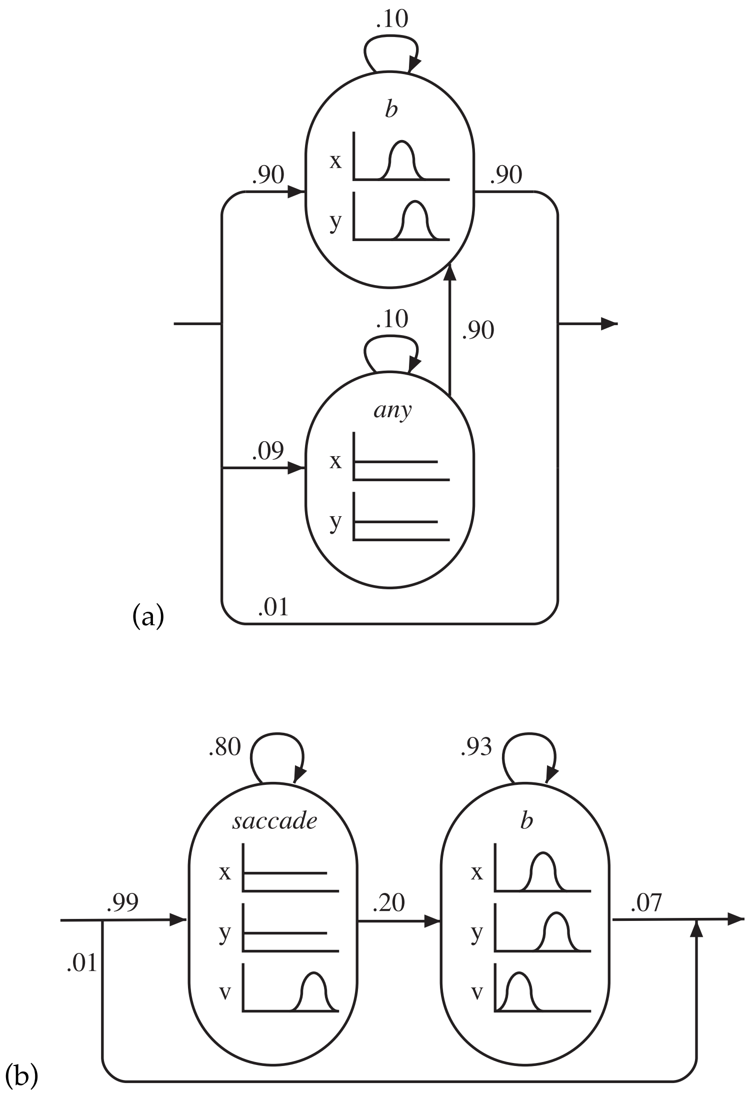


Figure 8.4: Fixation HMMs used with (a) Fixation-HMM and (b) Point.

Results with Tracing

The following analysis describes the results of tracing the eye-typing protocols with the various combinations of tracing methods, model grammars, and vocabularies. Results for the simple model represent the interpretations of Target-HMM; all of the target tracing methods produced very similar results except Target-TA, whose results were significantly worse. Results for the first- and second-order models represent the interpretations of Fixation-HMM and Point both with and without sequence matching. Results for the full model represent the interpretations of Fixation-HMM and Point strictly without sequence matching, since the full model maps protocols directly to vocabulary words.

Illustrative Protocols

We first look at how tracing handles the difficult protocols in Figure 8.2. Figure 8.5 shows the traced versions of these protocols as interpreted by Fixation-HMM; mapped fixations are labeled with a sequence number and the specific target to which they map, while incidental fixations are labeled only with a sequence number. In Figure 8.5(a), tracing correctly maps the off-center fixation 3 to Q and interprets fixations 4 and 6 as incidental. In Figure 8.5(b), it correctly interprets the off-center fixations 6 and 7; it also accounts for the skipped fixation of A by assigning fixation 4 to E but noting that the protocol has passed through a predicted fixation on A. In Figure 8.5(c), tracing ignores the many incidental fixations at the start of the protocol and maps only those that lie over or near the correct targets. Thus, by using the sequential information in the model grammars, tracing nicely handles the off-center, incidental, and skipped fixations in these protocols and generates the correct interpretations.

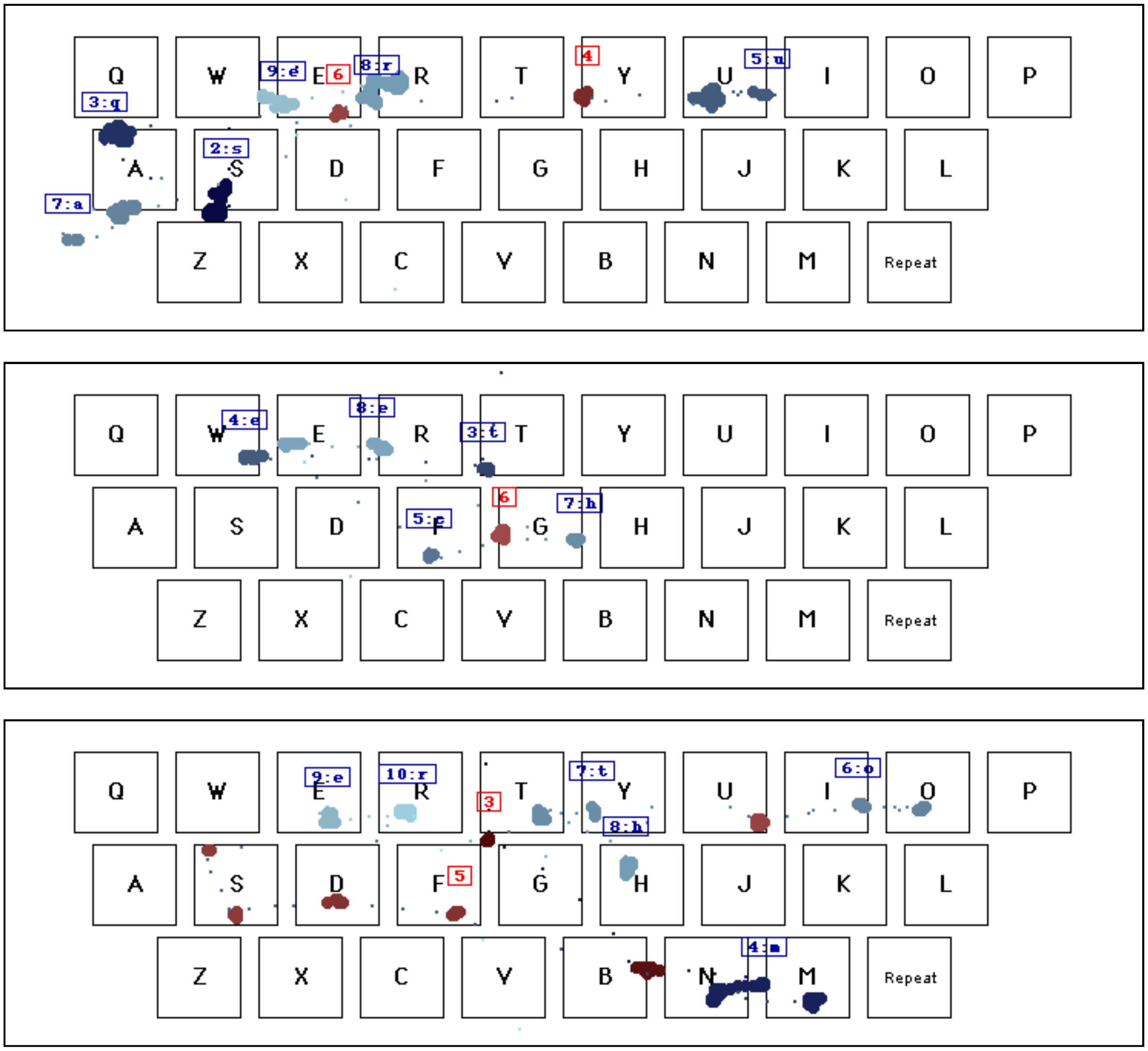


Figure 8.5: Protocols in Figure 8.2 traced with Fixation-HMM and interpreted correctly as the words (a) *SQUARE*, (b) *TEACHER*, and (c) *MOTHER*.

Interpretation Accuracy and Time

Interpretation accuracy measures how often the interpretation generated by tracing matches the given word to be typed in a particular trial. Table 8.4 shows the accuracy results for the various models and vocabularies without sequence matching. As discussed earlier, the simple model produces almost no correct interpretations. For the other three models, the models with more sequential information give the highest accuracy while those with less information give the lowest. The full model is highly accurate for all vocabularies and for both Fixation-HMM and Point; note that the Point full-model results for 1000 words are not included because of the inordinate amount of time required to compute them. The first-order and second-order models show much better accuracy for Fixation-HMM than for Point, with mediocre accuracy for the smallest vocabulary and poor accuracy for the largest.

Table 8.4: Interpretation accuracy without sequence matching.

Words	Simple	First-order		Second-order		Full	
		F-HMM	Point	F-HMM	Point	F-HMM	Point
12	.00	.43	.07	.77	.49	.99	.99
100	.00	.10	.02	.48	.05	.99	.98
1000	.00	.04	.01	.14	.01	.95	N/A

Because of an inevitable speed-accuracy tradeoff, it is important to examine interpretation time along with accuracy. Table 8.5 shows the time in milliseconds needed to interpret a single protocol. As expected, models with more information take more time to interpret protocols than those with less information. Most of the times are faster than real-time—that is, they are faster than the time taken to generate the protocols. The longest times for Fixation-HMM (full and second-order times for 1000 words) could realistically be sped up to near real-time with better HMM representations and search algorithms; the final chapter discusses this in more detail. Point requires much more time than Fixation-HMM to generate its interpretations, their times differing by an order of magnitude or more.

Table 8.5: Interpretation time without sequence matching, in ms.

Words	Simple	First-order		Second-order		Full	
		F-HMM	Point	F-HMM	Point	F-HMM	Point
12	48	57	992	98	2440	124	3262
100	50	83	1397	407	11701	867	31535
1000	49	81	1476	1005	25285	9104	N/A

By using sequence matching to map interpreted fixations to a vocabulary word, we can improve these results dramatically. Table 8.6 shows interpretation accuracy with sequence matching for the simple, first-order, and second-order grammars. As before, the models with more sequential information perform better than those with less information. However, accuracy overall is greatly improved: Accuracy for the simple, first-order, and second-order grammars with sequence matching approaches that for the full model (for which sequence matching is not needed). However, accuracy for the first three models degrades more rapidly for larger vocabularies than for the full model.

Table 8.6: Interpretation accuracy with sequence matching for simple, first-order, and second-order model grammars.

Words	Simple	First-order		Second-order		Full	
		F-HMM	Point	F-HMM	Point	F-HMM	Point
12	.89	.94	.94	.98	.99	.99	.99
100	.79	.83	.85	.91	.87	.99	.98
1000	.70	.74	.71	.75	.74	.95	N/A

Sequence matching adds to the time needed to generate interpretations, as shown in Table 8.7. While interpretation times for the first three models without sequence matching increase little for larger vocabularies, times with sequence matching increase more linearly with the number of words in the vocabularies. The full model without sequence matching still requires more time than the other models with sequence matching. Overall, the accuracy results in Tables 8.4 and 8.6 and the time results in Tables 8.5 and 8.7 nicely show the speed-accuracy tradeoff inherent in the various models. They also show that Fixation-HMM provides just as good accuracy as Point with significantly less time cost.

Table 8.7: Interpretation time with sequence matching for simple, first-order, and second-order model grammars, in ms.

Words	Simple	First-order		Second-order		Full	
		F-HMM	Point	F-HMM	Point	F-HMM	Point
12	58	70	1006	106	2446	124	3262
100	136	157	1483	483	11765	867	31535
1000	883	866	2326	1771	26093	9104	N/A

The interpretations shown above are "optimal" with respect to each model in the sense that they use the standard HMM decoding algorithm that maximizes the probability of each protocol given the model. It is interesting to explore the speed-accuracy tradeoffs when a sub-optimal search is employed—in particular, the beam decoding algorithm described in Appendix A. Beam decoding maintains only a small number of best interpretations at each point in time, eliminating the time needed to explore much less probable interpretations. However, because beam decoding prunes possible interpretations, it sometimes prunes away the optimal interpretation before fully exploring it. Tables 8.8 and 8.9 show the interpretation accuracy and time for Fixation-HMM and a full grammar for standard decoding and decoding with two beam sizes, 100 states and 1000 states. The 100-state beam maintains high accuracy for the 12-word vocabulary but drops off sharply for larger vocabularies, although its times are approximately 10 times faster than those for standard decoding for large vocabularies. The 1000-state beam provides reasonable accuracy for even the 1000-word vocabulary with significant time gains over standard decoding, reducing interpretation time from 9.1 seconds to 2.3 seconds for the 1000-word vocabulary. Thus beam decoding provides yet another dimension in which we can manipulate parameters to produce desired speed-accuracy tradeoffs.

Table 8.8: Interpretation accuracy for Fixation-HMM and full grammar with 100-state beam decoding, 1000-state beam decoding, and standard decoding.

Words	Decoding Method		
	Beam, 100	Beam, 1000	Standard
12	.99	.99	.99
100	.89	.99	.99
1000	.46	.88	.95

Table 8.9: Interpretation time, in ms, for Fixation-HMM and full grammar with 100-state beam decoding, 1000-state beam decoding, and standard decoding.

Words	Decoding Method		
	Beam, 100	Beam, 1000	Standard
12	121	137	124
100	227	927	867
1000	1198	2263	9104

Performance and Improvement

Because the eye-based interface was new to all subjects, it is interesting to measure their performance and any possible performance improvement over the duration of the experiment. We analyze subject performance and improvement with two measures: average typing time and average number of incidental fixations. Typing time is the average time needed to type one character, not including time needed to read the word and fixate the output box; the measure was computed using Fixation-HMM with the full model for the 12-word vocabulary. Overall, subjects required 821 ms per character on average to eye-type the words; the fastest subject averaged 430 ms, while the slowest averaged 1255 ms. These times are significantly faster than those reported for other eye-typing systems (e.g., Hutchinson et al., 1989: 85 min/page, Stampe & Reingold, 1995: 1870 ms/char). Subjects also showed a modest improvement over the two halves of the experiment, averaging 870 ms in the first half and 772 ms in the second half. Although this difference was not significant, F(1,6)=1.52, p>.2, only one subject exhibited an inexplicably higher second-half time; without this subject, the difference was very significant, F(1,5)=18.78, p<.01.

The number of incidental fixations is another measure that captures subject performance, since we expect more practiced typing to eliminate complex search procedures and thus produce fewer incidental fixations. Like typing time, this measure was computed using the interpretations of the full model for the 12-word vocabulary. On average, subjects reduced their incidental fixations dramatically during the experiment, producing 2.24 incidental fixations in the first half and 1.27 fixations in the second half. This difference is highly significant, F(1,7)=97.38, p<.001. These results, in conjunction with the typing time results, provide strong evidence that subjects can improve their performance in eye-based interfaces even after a small number of trials.

Comparing Eye Typing and Hand Typing

Another interesting measure compares subject performance in the eye-typing task with their performance in the hand-typing task. Naively, we might expect eye typing to correlate highly with hand typing, simply because better hand-typists have better working knowledge of the keyboard. However, it is far from clear whether this motor-oriented knowledge transfers to eye movements. In fact, we might even expect so-called “hunt-and-peck” hand-typists to perform better in eye typing because they often use their eyes to search the keyboard.

Eye-typing times as computed earlier were compared with hand-typing times as computed by the total trial time divided by the number of characters typed. Not surprisingly, hand typing was significantly faster than eye typing, F(1,6)=24.06, p<.01: Hand typing averaged 343 ms per character, while eye typing averaged 821 ms. Interestingly, eye-typing times were highly correlated with hand-typing times, *R*=.95. Thus, the expertise and skills involved in hand typing apparently transfer readily to eye typing.

Summary

By tracing the eye-typing protocols with four models over three vocabularies, we have shown that tracing can facilitate the interpretation of eye movements in an eye-based interface. The primary results of this study, interpretation accuracy and times, show a distinct speed-accuracy tradeoff with the various models: the full model provides the best but slowest performance, the simple model the worst but fastest performance, and the first-order and second-order models fall in between. The full model provides enough accuracy for a real system, while the other models are competitive when HMM-based tracing is augmented with sequence matching. Tracing thus allows interface designers and builders to choose an appropriate model given specific accuracy and time constraints, not only for eye-typing interfaces but other types of eye-based interfaces as well. For interfaces with few strategies or for which high accuracy is critical, a full model provides good results. For interfaces with many strategies or for which fast response is critical, a simple, first-order, or second-order model may be preferable. For the non-trivial (i.e., non-simple) models, Fixation-HMM stands out as the preferred method for good speed and accuracy.

Discussion

This study offers great promise for unimodal and multimodal interfaces that utilize eye-based input. Significant effort has gone into developing more accurate, less expensive, and less intrusive eye-tracking devices (e.g., Yang, Stiefelhagen, Meier, & Waibel, 1998). However, even an accurate, inexpensive, non-intrusive eye tracker is not sufficient for building effective eye-based interfaces; the systems would still need robust algorithms to interpret eye movements in the face of equipment and individual variability. This work helps to bridge the gap between a user's eye movements and his/her intentions. In addition, while eye movements can serve as the sole input modality for certain applications (e.g., assistive technology for disabled users), greater potential arises in the integration of eye movements with other input modalities—for instance, an interface in which eye movements provide pointer or cursor positioning while speech allows typing or directed commands.

This work has at least two important implications for future eye-based interfaces. First, user performance in eye-based interfaces can improve with even small amounts of practice. Analysis of the eye-typing data illustrates that, with repeated trials, users type words faster and produce fewer incidental fixations. In addition, cognitive and motor skills from similar domains (e.g., hand typing) can potentially transfer to eye-based interfaces. Second, removing limiting restrictions found in past eye-based interfaces—such as long dwell times and large spacing between visual targets—can significantly improve interface usability and flexibility. However, fewer restrictions inevitably lead to reduced accuracy, emphasizing the necessity for more powerful interpretation algorithms such as fixation tracing.

Distinguishing Thinking from Typing

While the tracing algorithms perform well in interpreting eye-movement protocols on a word-by-word basis, a more general eye-typing system must also determine word boundaries and distinguish between typing and "thinking"—that is, when a user might be pondering what to type next. The problem of determining word boundaries also arises in speech and handwriting recognition and can be quite complex to handle. For eye typing, we could simply require users to type spaces between words just as they might hand-type the words; the prototype interface used in the study has such a space bar embodied by the output box. However, the separation of thinking and typing can be quite difficult. Unlike speech and handwriting, eye movements are always "on"—the eyes must point somewhere, although they could be closed or directed off-screen. One possible method for identifying thinking might utilize a mismatch threshold for words such that, when the mismatch is above threshold, the system determines that the eye movements do not correspond to a word but rather random movement (see the discussion of the "unknown" strategy in the final chapter). However, there is no way to distinguish typing from thinking if the random thinking movements happen to correspond to real words. A real-world system may require the use of modal "typing" and "thinking" buttons that users explicitly activate when doing one or the other.

Relating Target Tracing to Spelling Suggestion

Target tracing in the eye-typing domain performs a very similar function to spelling suggestion in word processors: both take an observed word (e.g., a misspelled word in a word processor) and find a real dictionary word that best matches the observed word. However, the basic algorithm behind spelling suggestion is very different from the sequence-matching algorithm behind target tracing. Spelling suggestion typically uses one of two algorithms, the Soundex and Metaphone algorithms (Binstock & Rex, 1995). Both algorithms map a string of letters to a phonetic code (numbers and/or letters) such that words that sound alike map to the same code. When a word processor suggests possible spellings, it maps the base word to its phonetic code and generates a list of words with the same. Because such lists can be precomputed, spelling suggestion is typically very fast. It also works under the implicit assumption that people typically mistype words as words that would sound phonetically similar. While target tracing could use a similar algorithm, the most likely source of error in eye typing comes from misattributions of fixations to the wrong targets rather than misspellings. Nevertheless, it may be possible to precompute certain common mistakes in eye typing to speed up the interpretation process.

Chapter 9
Conclusions

Summary and Contributions

This thesis offers a number of important contributions to the fields of computer science, cognitive science, psychology, human-computer interaction, and related areas. Most importantly, it introduces and formalizes three tracing algorithms for interpreting eye movements with cognitive process models. It also collects and formalizes a number of existing and novel algorithms for identifying fixations in raw eye-movement protocols. The thesis evaluates the tracing and fixation-identification algorithms in three real-world application domains and demonstrates the benefits of the algorithms for each domain. All algorithms are implemented in a working system, EyeTracer, that allows for eye-movement protocol manipulation, visualization, and analysis.

Algorithms for Tracing Eye Movements

The first and most important contribution of this thesis is a class of tracing algorithms for interpreting eye movements with cognitive process models (Chapter 4). The thesis describes and formalizes three tracing algorithms: target tracing, fixation tracing, and point tracing. Target tracing is essentially an extension of an existing tracing algorithm that utilizes sequence matching to trace generic action protocols (Card, Moran, Newell, 1983). Fixation and point tracing are novel algorithms that apply hidden Markov models (HMMs) to tracing eye movements. Altogether, the three represent a powerful toolkit of algorithms that can be utilized for a variety of purposes, including protocol coding, model prototyping and refinement, model evaluation and comparison, and real-time inference of user intentions.

Evaluating Tracing Accuracy and Speed

To evaluate the accuracy and speed of the tracing algorithms, the thesis applies them to three domains—equation solving, reading, and eye-typing—and tests them directly and indirectly in the context of these domains. The constrained equation-solving and eye-typing studies provide the most direct tests of tracing accuracy. The constrained equation-solving study uses an instructed-strategy paradigm to give a "correct" interpretation of subject protocols. The interpretations of the tracing algorithms correspond very well to the "correct" interpretations; in fact, they are as accurate or more accurate than those of expert human coders. The eye-typing study is very similar in that the "correct" interpretation of a user's eye movements is known, namely the letters of the given word to be typed. Again the tracing algorithms provide accurate interpretations when compared to the "correct" interpretations. In both studies, point tracing and fixation tracing gave the best accuracy with target tracing close behind.

The unconstrained equation-solving and reading studies provide a more indirect evaluation of tracing accuracy. In the unconstrained equation-solving study, traced interpretations of protocols clear up subjects' intended fixation targets to better reveal when subjects perform intermediate computation. In the reading study, traced interpretations provide a better fit to expected results with respect to fixation durations and probabilities. Both studies demonstrate that the sequential information utilized by tracing helps sort out temporal information in the protocols, thus giving indirect evidence that the tracing algorithms generate sensible interpretations of the protocols. Amongst each other, the tracing algorithms differ little with respect to their ability to sort out temporal information.

The constrained equation-solving and eye-typing studies evaluate the speed of the tracing algorithms. The constrained equation-solving study shows that all the tracing algorithms interpret protocols approximately an order of magnitude faster than expert human coders. The eye-typing study shows that all algorithms can run in real-time, at least for smaller vocabularies of 1000 words or less. In addition, the study demonstrates that the algorithms differ greatly in terms of their speed-accuracy tradeoffs: target tracing is the fastest but least accurate algorithms, point tracing is the slowest but (generally) most accurate algorithm, and fixation tracing is almost as fast as target tracing and almost as accurate as point tracing.

Recommendations

Target tracing provides fast interpretations with reasonable accuracy and robustness to protocol noise. Its primary advantage is the ease in which a user can understand and implement the algorithm. Its accuracy, while not as good as the other algorithms, is likely adequate for many simple applications. Target tracing has three major disadvantages that hinder its use in practical applications. First and foremost, the algorithm cannot handle complex process models with many possible enumerated strategies. Hierarchical or circular process models with even a small number of rules can generate enough enumerated strategies to make target tracing infeasible. Second, target tracing cannot incorporate rule probabilities in any systematically meaningful way, forcing process models to consider every strategy equally likely. Third, target tracing produces a fairly coarse goodness-of-fit metric that can have only integer values. Nevertheless, target tracing is a more-than-adequate tracing algorithm that should serve well in simple applications.

Fixation tracing generates more accurate and robust interpretations than target tracing with only a minor loss of speed. Its speed can be improved with memoization of tracer HMMs and process models that generate sparse HMMs. In addition, unlike target tracing, fixation tracing can handle hierarchical and circular process models, overlapping and embedded target areas, and strategies of different probabilities. The difficulty of implementing the algorithm, its one major disadvantage, can be alleviated through the use of a public or commercial package for HMM construction and decoding. Overall, fixation tracing stands out as the most powerful and usable of the three tracing algorithms. Its balance of speed and accuracy, plus its flexibility in acceptable cognitive process models, makes it an excellent choice for many applications. In addition, as we discuss shortly, fixation tracing is the most amenable to future extensions such as the incorporation of fixation duration information and the generalization to dynamic task environments.

Point tracing is the least efficient of the tracing algorithms but also provides the best accuracy. Although it allows the cognitive process model to influence all aspects of tracing, its major difference with fixation tracing—the ability of the model to influence fixation identification—does not seem to have a major impact on tracing; in reality, fixation identification is often fairly robust, and thus sequential information in a process model helps very little in the process. In fact, this difference sometimes causes a problem with finding incidental fixations, because point tracing is more likely to identify them as saccades and not count them as incidental fixations. Thus, the extra predictive power of point tracing does not significantly improve its performance, and sometimes may even degrade the usefulness of its resulting traces.

Algorithms for Identifying Fixations

Another important contribution of the thesis is the collection and formalization of a set of useful algorithms for fixation identification (Chapter 3). While a number of researchers have developed such algorithms, only a few have compared the various algorithms (e.g., Karsh & Breitenbach, 1983) and to a very limited extent. The thesis gathers several existing and novel algorithms and collects them into a single presentation. I-VT (velocity-threshold identification), I-DT (dispersion-threshold identification), and I-TA (target-area identification) are formalizations of algorithms employed in previous work. I-HMM (hidden Markov model identification), like fixation and point tracing, is a novel application of HMMs to eye-movement data analysis. Like the tracing algorithms, these identification algorithms represent a useful toolkit of methods for different needs and applications.

Evaluating Identification Accuracy and Speed

The thesis evaluates the accuracy and speed of the four identification algorithms in the context of the application domains. The evaluation of identification accuracy arises primarily in the constrained equation-solving and eye-typing studies, where each algorithm is tested for its ability to facilitate target tracing and produce accurate tracing results. Both studies have similar results: I-HMM, I-VT, and I-DT give more accurate and robust interpretations than I-TA. The evaluation of identification speed, primarily in the constrained equation-solving study, shows that all four algorithms are fairly similar in terms of real-time efficiency, with I-HMM requiring slightly more time than the other three algorithms.

Recommendations

The choice of fixation-identification algorithm depends greatly on whether data analysis focuses on fixations or gazes (aggregations of consecutive fixations on the same target). For gaze analysis, I-HMM, I-VT, and I-DT all give good results; I-TA tends to miss more fixations and requires the specification of target areas, making it the least favorable choice. For fixation analysis, I-HMM and I-DT give the best results; I-TA cannot account for multiple fixations, and I-VT sometimes identifies too many fixations when point velocities hover near threshold. I-HMM may be the favorable choice because all its parameters can be estimated directly from data; however, I-DT may be preferred because it does not require implementation of HMM reestimation and decoding procedures.

Domain Applications

The third major contribution of this thesis is the application of the tracing and identification algorithms to three real-world domains: equation solving, reading, and eye typing (Chapters 6-8). As described above, the studies of these domains help evaluate the accuracy and speed of the algorithms. However, they also illustrate how the algorithms, and tracing more generally, can greatly benefit work in each domain. The three domains are highly complementary, and thus together allow for a comprehensive demonstration of the power and usefulness of the tracing algorithms.

Equation Solving

The equation-solving studies demonstrate how tracing assists in coding experimental eye-movement protocols and in developing cognitive process models (Chapter 6). The constrained study uses an "instructed-strategy" paradigm in which subjects are given specific problem-solving strategies and asked to utilize these strategies in the equation-solving task. As is often done with verbal protocols, the resulting eye-movement protocols can be coded by categorizing each protocol as one of the given strategies. Results show that the tracing algorithms can code protocols as accurately or more accurately than expert human coders, but in much less time. These results manifest the great promise for these and similar tracing algorithms to code eye-movement protocols and thus facilitate higher-level analyses.

The unconstrained study focuses on modeling subject behavior in the unconstrained task, in which subjects could solve problems in whatever way they choose. The study first performs an exploratory analysis on the data set using frequency trees and protocol visualization to identify common strategies. Starting with a prototype model grammar that embodies these strategies, the study then performs three iterations of trace-based protocol analysis (Ritter & Larkin, 1994), where each iteration comprises: tracing the data set with the model, evaluating the model and identifying model-data mismatches, and refining the model accordingly. After the three iterations, the study shows how the traced data help understanding of subject behavior, especially with respect to temporal aspects of behavior (e.g., gaze durations). The study also takes the final model grammar and from it derives an ACT-R/PM (Anderson & Lebiere, 1998) cognitive model of subject behavior. This model predicts both the cognitive steps involved in the task and the time of each observable action (i.e., gaze or keystroke). Thus, the unconstrained study illustrates the use of the eye-movement tracing algorithms in trace-based analysis for the creation and refinement of a cognitive process model.

Reading

The reading study demonstrates how tracing facilitates the evaluation and comparison of existing cognitive process models (Chapter 7). The study focuses on two computational process models of reading, E-Z Reader 3 and 5 (Reichle et al., 1998). It first derives first- and second-order model grammars from the E-Z Reader models that represent their first- and second-order transition characteristics, respectively. With these grammars, the study compares the E-Z Reader models and finds that the qualitatively better model, E-Z Reader 5, produces a better quantitative fit to collected data. In addition, the study shows that, like in the unconstrained equation-solving study, tracing cleans up temporal aspects of the data (i.e., fixation durations), producing results that better conform to earlier reported results. The study gives a good illustration of how the tracing algorithms help compare cognitive process models and sort out temporal information by means of sequential information.

Eye Typing

The eye-typing study illustrates how tracing assists in designing and implementing eye-based user interfaces (Chapter 8). The experiment in the study uses a prototype eye-typing interface in which users can type words by looking at letters on an on-screen keypad. This interface differs from previous interfaces in that it eliminates two major restrictions imposed by these interfaces: large spacing between letters and long dwell times to actuate a typed letter. A naive algorithm that simply maps fixations to the nearest letter does an extremely poor job at interpreting user eye movements. However, the tracing algorithms, for different models and vocabularies, provide reasonable to excellent tracing accuracy with corresponding tradeoffs in tracing speed. Overall, the study shows that the tracing algorithms help compensate for the difficult of eliminating the two common interface restrictions. It also shows that the algorithms can be used for real-time interpretation in eye-based interfaces or any interfaces that may benefit from tracing eye movements, such as intelligent tutoring systems.

The EyeTracer System

A fourth contribution of this thesis is the EyeTracer system, a development environment in which a user can manipulate and analyze eye-movement protocols (Chapter 5). EyeTracer embodies all the algorithms described in the thesis and provides both a testbed for the algorithms and a working system for anyone interested in tracing eye-movement protocols. The system has a menu-driven interface in which a user can view and analyze protocols for both exploratory and confirmatory data analysis. The system is also modular such that a user can simply enter information specific to a particular data set and run the algorithms on the data set with no further implementation. EyeTracer runs in the Macintosh Common Lisp environment and is publicly available at the thesis web site.[11]

Extensions and Future Work

The work in this thesis can be extended in a number of ways. Several possible extensions involve removing one or more of the basic assumptions made in the first chapter, particularly the assumptions that the task screen is static and that relevant target areas can be easily specified. Other extensions further exploit the power of hidden Markov models by encoding probabilistic duration distributions and by reestimating tracer model parameters.

---

[11] http://www.cs.cmu.edu/~dario/TH99/

While a detailed specification of such extensions is beyond the scope of this thesis, the following exposition gives an overview of the more promising directions for future work in this area.

Tracing with Fixation Durations

The tracing algorithms proposed in this thesis utilize only the locations of fixations and the sequence in which they occur. Additional information present in an eye-movement protocol, particularly fixation durations, can potentially facilitate interpretation of the protocol. For instance, in the equation-solving domain, our model of behavior in the task assumes that people compute intermediate results as soon as possible; thus, the model predicts that fixations during which computation takes place have longer durations than those during which no computation can occur. As another example, in the eye-typing domain, we may suspect that incidental fixations are generally shorter than actuating fixations and build this into our model of user behavior. The distribution of possible fixation durations can be estimated or derived from a cognitive process model that can predict durations, such as the ACT-R/PM model developed in the unconstrained equation-solving study. By utilizing fixation durations in tracing, we incorporate more information into the tracing process and thus allow for more powerful and more robust interpretations.

Fixation tracing is the most amenable of the tracing algorithms to such an extension. Let us assume that fixation identification produces a sequence of fixations of the form <*x*, *y*, *d*>, where *x* and *y* represent the fixation's spatial location and *d* its duration. When using spatial information alone, the fixation HMMs used in fixation tracing contain probabilistic distributions for *x* and *y* describing the expected fixation location for a particular target area. To incorporate duration information, we simply add a probabilistic distribution for *d* that describes the expected fixation duration, as shown in Figure 9.1. The distribution need not be normal; in fact, some research has shown that fixation durations are distributed approximately as an exponential distribution after a minimum threshold (Harris et al., 1988) similar to the distribution for *d* in Figure 9.1. During tracer model decoding, fixations with a duration closer to the expected duration match with a higher probability and thus generate more likely traces.

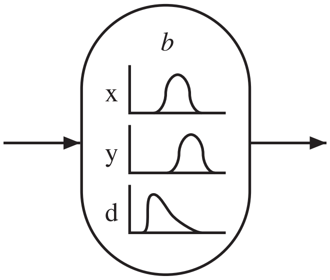


Figure 9.1: Fixation HMM that incorporates a fixation-duration distribution *d*.

While target tracing and point tracing can incorporate fixation durations in their algorithms, the incorporation is more cumbersome than for fixation tracing. For target tracing, the sequence-matching algorithm needs to be altered such that duration mismatches are counted in the total mismatch metric. This change requires a (somewhat arbitrary) conversion of duration mismatch in units of time to mismatch in units of insertions, deletions, and substitutions. For point tracing, fixation durations have an implicit exponential distribution described by the value of a fixation state's self-transition. However, there is no minimum threshold in this distribution, making extremely short fixations very likely. To better model fixation durations, point tracing would require the use of "time-duration modeling"—that is, explicit distributions that encode the number of self-transitions for a particular state. Time-duration modeling has been used in speech recognition with limited success (see Rabiner, 1989) but seriously degrades the efficiency of decoding; given that point tracing is by far the slowest of the tracing algorithms as is, such a change would generally be infeasible.

Tracing in Dynamic Task Environments

Another limitation of the proposed tracing algorithms is the assumption that the task environment is *static*, with few if any changes occurring in the target areas. Clearly there are a number of interesting tasks that do not obey this assumption. These so-called *dynamic* tasks allow for targets to appear and disappear at various times, and also for targets to move about the screen; for example, a radar-tracking task involves monitoring a number of friendly and/or enemy aircraft as they appear on radar and move about the range of view. For the sake of illustration, we consider two types of targets in dynamic tasks: a stationary target that appears in mid-task and disappears after some time, and a moving target that also appears and disappears. We now briefly discuss possible extensions to fixation tracing that allow for tracing of stationary and moving dynamic targets. Such extensions would enable systems that infer user intent in dynamic tasks (e.g., Pentland & Liu, 1999) to incorporate eye movements and thus improve performance.

Stationary dynamic targets, as we define them, appear at a fixed location x and y, remain at this location for some time, and finally disappear from view. One example of such a target is a help window in an intelligent tutoring system, which pops up when the user asks for help and disappears upon closing. To trace a stationary dynamic target, we can augment the fixation HMM in fixation tracing in a way similar to that for fixation durations: we add a distribution for t to the HMM state that represents the probability of fixating the target at any given time. Figure 9.2(a) shows a sample fixation HMM with a distribution for t that starts at the time at which the target appears and trails off to zero when the target disappears; this distribution incorporates the assumption that the target is more likely to be fixated soon after it appears. If we assume that fixations have the form <*x*, *y*, *t*>, where *t* is the fixation onset time, fixation tracing uses the onset time information to influence interpretation and increase the probability of fixations on the target at expected times.

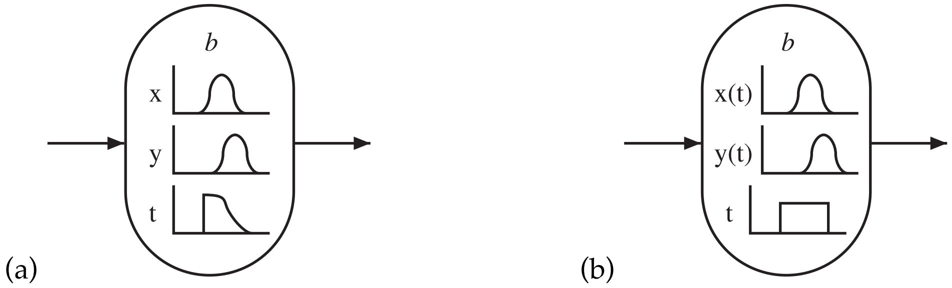


Figure 9.2: Fixation HMMs that incorporate a fixation-time distribution *t* for a (a) stationary target and (b) moving target.

Moving dynamic targets appear and disappear like stationary targets, but their spatial location changes with time—for instance, an aircraft radar blip in the radar-tracking task. Fixation tracing can model moving dynamic targets with a slight modification of the stationary target HMM: we simply make the probability distributions *x* and *y* for dependent on time *t*, as shown in Figure 9.2(b). During decoding, when the probability of a given fixation <*x*, *y*, *t*> is computed, the probabilities of the observed *x* and *y* are computed as functions of *t*. Also, unlike the stationary-target HMM in Figure 9.2(a), this moving-target HMM assumes a uniform probability of fixating the target at any time during its presence. Of course, a moving target may elicit smooth rather than saccadic eye movements; smooth eye movements would require some modification of the fixation-identification algorithms to classify smooth movement or, in some sense, "moving" fixations. Nevertheless, the two HMMs in Figure 9.2 illustrate that fixation tracing is very amenable to future extensions that handle tracing in dynamic task environments.

Multimodal Tracing

While eye movements alone reveal a great deal about a person's thoughts, eye movements in conjunction with other types of data can provide even fuller and more robust information. Multimodal tracing, as we describe it, is the simultaneous tracing of multiple types of data from different modalities, such as eye movements, mouse movements, keystrokes, speech, and gesture. While serious effort has gone into interpreting data from each modality alone, much less work has explored how to generate simultaneous interpretation of multiple modalities such that one can influence interpretation of another. Multimodal tracing would benefit almost any field that benefits from unimodal tracing, such as user interfaces and experimental data analysis.

While this thesis does not address multimodal tracing directly, it does implicitly suggest how to interpret eye movements along with other types of data. In their use of hidden Markov models, fixation and point tracing illustrate a strong conceptual link between eye movements and other data such as speech and handwriting: all these modalities can utilize stochastic models to map observed data to predicted action sequences, thus overcoming individual variability and sampling noise. The common method of interpretation greatly facilitates the incorporation of eye-movement data into future work in multimodal tracing; for instance, we could imagine a multimodal HMM whose states predict expected observations of both speech and eye movements. The problems that arise in building robust multimodal tracing algorithms are far from trivial, but this thesis represents an important step toward this goal.

Tracing Smooth Eye Movements

This thesis has solely addressed the problem of tracing saccadic eye movements, in which the eye moves ballistically from one fixation point to another. Another type of eye movement, smooth eye movement, can also occur even in constrained computer tasks. In smooth eye movements, the eye follows a moving target with a (for the most part) constant velocity—for instance, the movement required to track a moving blip on a radar screen. Smooth eye movements have been studied to a somewhat lesser extent than saccadic eye movements, but the basic mechanisms are reasonably well understood. Only Kowler, Martins, & Pavel (1984) have investigated algorithms for tracing smooth movements, but they used only two-state HMMs to distinguish movement to the left or right. A more general algorithm for tracing smooth movements would better help to understand behavior in tasks where smooth movement is possible, especially in classifying smooth vs. saccadic eye movement.

Tracing smooth eye movements would require some extension of the proposed fixation and point tracing algorithms. Like saccades, smooth movements rise to some non-zero velocity and maintain this velocity for some distance. However, smooth movements are different in two ways: they maintain the velocity for longer periods of time than saccades and do not reach as high a peak velocity. An HMM that traces smooth movement can include velocity distributions centered around typical smooth velocities. The HMM may also force a protocol to stay in smooth-movement states for longer times using, say, time-duration modeling. If the HMM could be given some knowledge of where moving targets appear on-screen, this knowledge can be utilized to better identify smooth movement.

Training of Tracer Models

Throughout the thesis we have utilized hidden Markov models as a tool for decoding, or interpreting, eye movements. However, the thesis largely ignores another significant benefit of

HMMs: the ability to train HMMs by reestimating parameter values that maximize the fit of the model to a given data set. Other fields that use HMMs, such as speech and handwriting recognition, have abundantly and successfully employed reestimation to train models of behavior for these data. This training has focused primarily on learning the parameters of low-level models of "atomic" behavioral components, such as phonemes in speech and strokes in handwriting. It also has explored how training can learn higher-level information, particularly language-based information such as word frequency and co-occurrence.

This thesis employs reestimation only in the context of learning the transition and observation parameters of the velocity HMM for I-HMM—similar to the training of "atomic" components in speech and handwriting. However, there are other ways in which reestimation could be used for higher-level training. For instance, reestimation of a tracer model could shift the x and y distributions within target states in cases where people often utilize parafoveal encoding. Also, reestimation could learn the probabilities of transitions between states, thus estimating the probabilities of the various strategies embodied by the tracer model. One problem that arises with reestimating the tracer model is the interpretation of the newly-estimated parameters with respect to the original model; because the tracer models are often derived from more comprehensive process models (e.g., an ACT-R/PM model), we would like to port estimated parameter values from the tracer model back to the original model. This task depends highly on the representation of the original model and may be quite complex. In any case, exploring how to reestimate tracer models and how to make use of estimated values offers a number of avenues for future work.

Tracing Confidence and the "Unknown" Strategy

The tracing algorithms as described all determine a single best strategy from a set of possible strategies. However, the algorithms provide no estimate as to the confidence of their interpretations. For instance, it may be convenient to gauge how well the best-matching strategy fits a protocol as compared to the second-best matching strategy, where larger differences in match (or mismatch) would indicate more confident interpretations. Also, we might allow tracing to decide that a protocol does not fit any of the given possible strategies, thus classifying the protocol as the "unknown" strategy. The most straightforward way to implement such an extension in the proposed algorithms is by assigning a mismatch threshold: a protocol with a mismatch below threshold is interpreted as the best-matching strategy, while one with a mismatch above threshold is interpreted as the "unknown" strategy. More systematic and robust methods for measuring tracing confidence could potentially be borrowed from similar techniques in speech recognition.

Other Extensions

In addition to the major extensions proposed above, there are several more minor or peripheral extensions that warrant mention. One involves applying the tracing algorithms to adjust the calibration of eye-tracking equipment when there is a constant shift <*dx*, *dy*> between the estimated and true point-of-regard locations. By trying different-valued shifts <*dx*, *dy*> and evaluating each shifted protocol with tracing, we can find the shift that produces the best model fit and re-adjust calibration accordingly. This technique has been implemented in the EyeTracer system with good results. Another possible extension generalizes the tracing algorithms to domains in which relevant target areas cannot be easily identified—for instance, picture scanning (e.g., Noton & Stark, 1971). Some work has explored how to find meaningful target areas (e.g., Stark & Ellis, 1981), and it is plausible that tracer model reestimation could assist in this task. A final important extension involves modifying the current off-line tracing algorithms to more on-line algorithms. Although domains such as eye typing show how tracing can simulate on-line interpretation, the system must still segment the input stream and pause to interpret it. A more on-line algorithm would help to produce smoother performance when real-time interpretation is needed and may scale better for more complex process models.

Appendix A

Hidden Markov Models

Hidden Markov models (HMMs) have received burgeoning attention in the past few decades as a rich tool for modeling real-world signals. Signals, simply put, are sequences of observations representing a stream of discrete or continuous output. It is often useful to represent signals in terms of a statistical model for the purposes of prediction, recognition, or even theoretical understanding of the signal source. HMMs provide a powerful and flexible way of modeling signals and building practical systems that rely on these models. Because eye movements in essence represent a signal themselves, HMMs can readily apply to eye movements just as they apply to similar signals such as speech and handwriting. The following exposition presents a brief overview of the theory and formalization of HMMs as is needed for tracing eye movements in the context of this thesis. For a more complete presentation of HMMs and their applications to speech in particular, Rabiner (1989) provides an excellent tutorial on HMMs and Huang, Ariki, and Jack (1990) provide a deeper analysis of their theory and implementation.

Hidden Markov models represent an extension of simple, or observable, Markov models. A simple Markov model has three components: a set of states, an observation (i.e., physical event) associated with each state, and a set of transition probabilities between states. This model represents "fully observable" Markov processes in that there is a direct and obvious mapping between observations and states. HMMs extend Markov models by allowing each state to produce different observations according to probabilistic distributions. HMMs thus represent "hidden" Markov processes in that there may be numerous mappings between observations and states—that is, the state sequence is "hidden" from observation.

Let us briefly look at two examples of hidden Markov models based on Rabiner (1989). Consider a situation in which a person has two biased coins, one that gives "heads" (H) 70% of the time, and one that gives "tails" (T) 70% of the time. Now assume that the person stands behind a curtain and flips one of the coins, reporting as observations only whether the coin gives H or T. We would like to determine, based on the sequence of observations H or T, which coin is being used for each flip. To do this, we create an HMM with two states, shown in Figure A.1(a): one that represents a flip of the first coin, and another that represents a flip of the second coin. Each state contains the probabilities for generating the two possible observations, as dictated by the biases of coins; the probability distributions are "discrete" in that they allow for two different discrete observations. The transition probabilities between the two states model the likelihood of switching coins versus flipping the same coin again; these probabilities indicate an 90% chance of re-flipping and a 10% chance of switching.

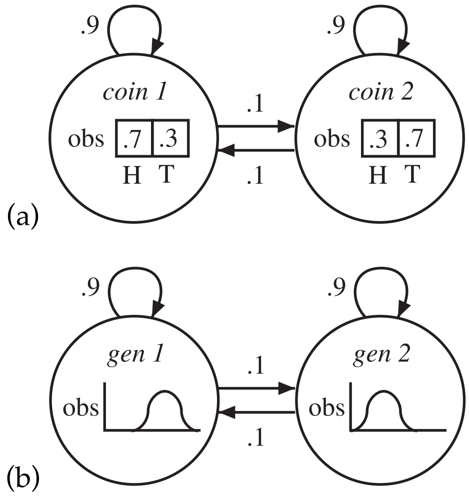


Figure A.1: Sample HMMs with (a) discrete and (b) continuous observation probability distributions.

While this type of discrete HMM (i.e., an HMM with discrete observation probability distributions) is often useful, it is sometimes better to represent observations as continuous distributions. Consider a situation slightly different from that above in which the person randomly generates real numbers between 0 and 1. The person uses two generators, one for numbers closer to 1 and one for numbers closer to 0, and reports observations as the real numbers generated. Figure A.1(b) shows an HMM that nicely models this situation, with continuous distributions that describe the different generators. Like before, the transition probabilities represent the likelihood of using the same generator or switching to the other generator.

The power of hidden Markov models comes primarily from three operations: decoding, evaluation, and reestimation. Assume we are given an HMM λ and a sequence of observations Ω. Decoding finds the state sequence that represents the best interpretation of the observations given the model. Evaluation finds the probability $\Pr(\Omega \mid \lambda)$ of the observations given the model. Reestimation adjusts the parameters of the model to maximize $\Pr(\Omega \mid \lambda)$. We now formalize a representation of the continuous HMM and describe several algorithms for performing these three operations given the model and observations. The descriptions of the algorithms include only the equations and basic steps needed to implement the algorithms; please refer to Rabiner (1989) for fuller interpretations of the variables and Huang, Ariki, and Jack (1990) for proofs of correctness.

Representation

A continuous HMM λ can be defined as the set of parameters $\{N, M, \Pi, A, B\}$ where these parameters represent the following:

- N = number of states in the model

- M = number of feature values in each state

- Π = prior probabilities $\{ \pi_i \}$ for all states i
  π_i = probability of starting in state i

- A = transition probabilities $\{ a_{ij} \}$ for all states i, j
  a_{ij} = probability of making a transition from state i to state j

- B = observation parameters $\{ \mu_i , \mathbf{U}_i \}$ for all states i
  μ_i = vector containing the Gaussian means for M features in state i
  $\mathbf{U}_i$ = matrix containing the Gaussian covariance matrix for M features in state i

The observation sequences Ω can be defined as follows:

- Ω = set of K observation sequences O^k
  O^k = sequence of T_k observations $\mathbf{o}_0^k \cdots \mathbf{o}_{T_k-1}^k$
  $\mathbf{o}_i^k$ = observation comprising M continuous feature values $o_{i,0}^k \cdots o_{i,M-1}^k$

Finally, the observation probability of a given observation $\mathbf{o}$ in a given state i can be computed with respect to the model's observation parameters as

$$b_i(\mathbf{o}) = \mathrm{G}(\mathbf{o}, \mathbf{m}_i, \mathbf{U}_i)$$

where G denotes the Gaussian distribution with the given mean and covariance matrix.

Decoding

Decoding an observation sequence O with respect to a model λ involves finding the observation sequence's most likely state sequence S—that is, finding the state sequence S that maximizes $\Pr(O,S \mid \lambda)$. The state sequence, or path, S thus represents the optimal interpretation of the data given the model with this particular optimality criteria. The path S can be found efficiently using a dynamic programming algorithm called the Viterbi algorithm. The Viterbi algorithm uses two variables: $\delta_t(i)$, representing the probability of the best path to state i at time t; and $\psi_t(i)$, the index of the previous state in the best path. Both variables depend on time t and state i. First, we initialize the variables for the first time step where t=0:

$$\delta_0(i) = \pi_i b_i(\mathbf{o}_0) \qquad 0 <= i < N$$

$$\psi_0(i) = 0 \qquad 0 <= i < N$$

Next, we perform an inductive step over the two variables in which we successively update the variables for each time t:

$$\delta_{t+1}(j) = \max_{i=0}^{N-1}\left[\delta_t(i)a_{ij}\right]b_j(\mathbf{o}_{t+1}) \qquad \begin{array}{c} 0 <= t < T-1 \\ 0 <= j < N \end{array}$$

$$\psi_{t+1}(j) = \arg\max_{i=0}^{N-1}\left[\delta_t(i)a_{ij}\right] \qquad \begin{array}{c} 0 <= t < T-1 \\ 0 <= j < N \end{array}$$

Finally, we compute the optimal state sequence $S = s_0 \cdots s_{T-1}$ and the probability $\Pr(O,S \mid \lambda)$ of this sequence as follows:

$$s_{T-1} = \arg\max_{i=0}^{N-1}\left[\delta_{T-1}(i)\right]$$

$$s_t = \psi_{t+1}(s_{t+1}) \qquad 0 <= t < T-1$$

$$\Pr(O,S \mid \lambda) = \max_{i=0}^{N-1}\left[\delta_{T-1}(i)\right]$$

The standard Viterbi decoding method described above is guaranteed to find the optimal state sequence that maximizes Pr(O,S | λ). However, it is often the case that only a few state sequences have any significant probability while most have extremely small probabilities. A variant of standard decoding, *beam* decoding, utilizes this fact to improve the computational efficiency of the Viterbi algorithm. Beam decoding maintains a "beam" of the n most probable states at each time step, where n is called the *beam size*. More formally, at each time t, beam decoding maintains $\delta_t(i)$ and $\psi_t(i)$ for only the n states with the highest $\delta_t(i)$. While states with very low probabilities may "catch up" to be included in the best state sequence, beam decoding makes the assumption that this happens infrequently. Thus, beam decoding trades off some of the accuracy of standard decoding for improved performance. Empirical tests have shown that beam decoding can provide near optimal results in significantly less time (e.g., Huang, Ariki, & Jack, 1990; this thesis, Chapter 8).

Evaluation

Evaluation of an HMM is defined as the probability of the observation sequences given the model, or Pr(Ω | λ). This probability can be computed using the so-called "forward-backward" algorithm. The algorithm uses three variables: a *forward variable* $\alpha_t^k(i)$, a scaling constant c_t^k and a scaled forward variable $\hat{\alpha}_t^k(i)$. The forward variables depend on the observation sequence index k, the time t, and the state i, while the scaling constant depends only on k and t. First, we initialize the variables for t=0:

$$\alpha_0^k(i) = \pi_i b_i(\mathbf{o}_0^k) \qquad \begin{array}{c} 0 <= k < K \\ 0 <= i < N \end{array}$$

$$c_0^k = \frac{1}{\sum_{i=0}^{N-1} \alpha_0^k(i)} \qquad 0 <= k < K$$

$$\hat{\alpha}_0^k(i) = c_0^k \alpha_0^k(i) \qquad \begin{array}{c} 0 <= k < K \\ 0 <= i < N \end{array}$$

Next, we successively update the variables for each time t:

$$\alpha_{t+1}^k(j) = \left[\sum_{t=0}^{N-1} \hat{\alpha}_t^k(i) a_{ij} \right] b_j(\mathbf{o}_{t+1}^k) \qquad \begin{array}{c} 0 <= k < K \\ 0 <= t < T_m - 1 \\ 0 <= j < N \end{array}$$

$$c_{t+1}^{k} = \frac{1}{\sum_{i=0}^{N-1} \alpha_{t+1}^{k}(i)} \qquad \begin{gathered} 0 <= k < K \\ 0 <= t < T_m-1 \end{gathered}$$

$$\hat{\alpha}_{t+1}^{k}(j) = c_{t+1}^{k} \alpha_{t+1}^{k}(j) \qquad \begin{gathered} 0 <= k < K \\ 0 <= t < T_m-1 \\ 0 <= j < N \end{gathered}$$

Finally, we compute the probability $\Pr(\Omega \mid \lambda)$ as the product of c_t^k for all k and t. However, because this probability is often extremely small (i.e., too small for standard computer hardware), we often instead compute the log probability as the sum of c_t^k:

$$\Pr(\Omega \mid \lambda) = \prod_{k=0}^{K-1} \prod_{t=0}^{T_k-1} c_t^k$$

$$\log \Pr(\Omega \mid \lambda) = \sum_{k=0}^{K-1} \sum_{t=0}^{T_k-1} \log c_t$$

Reestimation

Reestimation involves estimating new values for the parameters of the model λ so that they provide a better fit to the observed sequences Ω—in other words, so that they improve $\Pr(\Omega \mid \lambda)$. The most common reestimation algorithm is the Baum-Welch algorithm, which guarantees that $\Pr(\Omega \mid \lambda)$ never decreases with each reestimation. The algorithm employs the variables used in evaluation plus a number of new variables to reestimate the values of the parameters Π, A, B. First, we compute a *backward variable* in a manner similar to the computation of the forward variable in evaluation:

$$\hat{\beta}_{T-1}^{k}(i) = c_{T-1}^{k} \qquad \begin{gathered} 0 <= k < K \\ 0 <= i < N \end{gathered}$$

$$\hat{\beta}_{t}^{k}(i) = c_{t}^{k} \left[\sum_{j=0}^{N-1} a_{ij} b_j(\mathbf{o}_{t+1}^{k}) \beta_{t+1}^{k}(j) \right] \qquad \begin{gathered} 0 <= k < K \\ 0 <= t < T_k-1 \\ 0 <= i < N \end{gathered}$$

We then compute a number of helper variables, the meanings of which are fully described in the earlier-referenced citations:

$$\xi_t^k(i,j) = \hat{\alpha}_t^k(i) a_{ij} b_j(\mathbf{o}_{t+1}^k) \hat{\beta}_{t+1}^k(j) \qquad \begin{array}{l} 0 <= k < K \\ 0 <= t < T_k \\ 0 <= i, j < N \end{array}$$

$$\gamma_t^k(i) = \frac{\hat{\alpha}_t^k(i) \hat{\beta}_t^k(i)}{c_t^k} \qquad \begin{array}{l} 0 <= k < K \\ 0 <= t < T_k \\ 0 <= i < N \end{array}$$

Finally, we compute the new values for the various HMM parameters as follows:

$$\pi_i' = \frac{\sum_{k=0}^{K-1} \gamma_0^k(i)}{K} \qquad 0 <= i < N$$

$$a_{ij}' = \frac{\sum_{k=0}^{K-1} \sum_{t=0}^{T_k-2} \xi_t^k(i,j)}{\sum_{k=0}^{K-1} \sum_{t=0}^{T_k-2} \gamma_t^k(i)} \qquad 0 <= i, j < N$$

$$\mu_i' = \frac{\sum_{k=0}^{K-1} \sum_{t=0}^{T_k-1} \gamma_t^k(i) \cdot \mathbf{o}_t^k}{\sum_{k=0}^{K-1} \sum_{t=0}^{T_k-1} \gamma_t^k(i)} \qquad 0 <= i < N$$

$$\mathbf{U}_i' = \frac{\sum_{k=0}^{K-1} \sum_{t=0}^{T_k-1} \gamma_t^k(i) \cdot (\mathbf{o}_t^k - \mu_t^k)(\mathbf{o}_t^k - \mu_t^k)'}{\sum_{k=0}^{K-1} \sum_{t=0}^{T_k-1} \gamma_t^k(i)} \qquad 0 <= i < N$$

Note that the computations for μ_i and $\mathbf{U}_i$ require vector and matrix computation, respectively, where ′ denotes vector transpose.

Appendix B

Sequence Matching

Sequence matching is the process of determining the best match, or alignment, between two sequences of arbitrary symbols. The best match is typically defined as the match that minimizes the number of insertions, deletions, and substitutions needed to transform one sequence to the other. Sequence matching can be performed efficiently using a common dynamic-programming algorithm (Card, Moran, & Newell, 1983; Kruskal, 1983). This appendix provides pseudocode for the sequence-matching algorithm; please refer to the above references for a more detailed explanation of the algorithm and numerous possible variations.

The following pseudocode includes two main functions. The first main function **`sequence-match`** computes the sequence match between two sequences $A = A[0]...A[M-1]$ and $B = B[0]...B[N-1]$. The helper function **`score-matrix`** builds up a score matrix that describes match scores between the sequences up to various positions in the sequences. The helper function **`find-matches`** takes the score matrix and extracts the best match, represented as a list for each sequence in which matching elements align with each other and non-matching elements align with the special symbol *nil*. The second main function **`sequence-best-match`** finds the best match for a particular sequence A given a set of candidate sequences $Bs = Bs[0]\ldots Bs[K\text{-}1]$. The function **`mismatch-penalty`** helps this main function favor sequence pairs with earlier matches. All functions allow for the possibility of a special symbol *any* that can match any arbitrary symbol.

```
score-matrix (A, B)
  M = length(A), N = length(B)
  create matrix S of size (M+1)x(N+1) and initialize elements to 0
  for m = 1 to M
```

```
      for n from 1 to N
         if (a[m–1] = b[n–1] or a[m–1] = any or b[n–1] = any)
            S[m,n] = 1 + S[m-1,n-1]
         else
            S[m,n] = max (S[m,n-1], S[m-1,n], S[m-1,n-1])
   return S

find-matches (A, B, S)
   M = length(A), N = length(B)
   mismatch = M – S[M,N]
   A-match = nil, B-match = nil
   m = M , n = N
   do until (m = 0 and n = 0)
      if (m > 0 and (n = 0 or S[m-1,n] > S[m-1,n-1]))
         push a[m–1] onto A-match
         push nil onto B-match
         decrement m
      else if (n > 0 and (m = 0 or S[m,n-1] > S[m-1,n-1]))
         push nil onto A-match
         push b[n–1] onto B-match
         decrement n
      else
         push a[m–1] onto A-match
         push b[n–1] onto B-match
         decrement m
         decrement n
   return mismatch, A-match, B-match

sequence-match (A, B)
   S = score-matrix (A, B)
   return find-matches (A, B, S)

mismatch-penalty (A-match, B-match)
   penalty = 0
   for i = 0 to length(A-match)–1
      if (A-match[i] = B-match[i])
         penalty = penalty + 2^i
      else if (A-match[i] = any or B-match[i] = any)
```

```
      penalty = penalty + 1 + 2^i
   return penalty

sequence-best-match (A, Bs)
   K = length(Bs)
   compute matches = sequence-match (A, B) for each B in Bs
   best-mismatch = minimum mismatch of all matches
   compute best-matches = all matches with the best-mismatch
   compute mismatch-penalty (A-match, B-match) for all best-matches
   best-match = match in best-matches with the minimum mismatch penalty
   return best-match
```

References

Aaltonen, A., Hyrskykari, A., & Räihä, K. (1998). 101 spots, or how do users read menus?. In Human Factors in Computing Systems: CHI 98 Conference Proceedings (pp. 132-139). New York: ACM Press.

Alpern, M. (1962). Type of movement. In H. Davson (Ed.), *The Eye*, Vol. 3. (pp. 63-151). New York: Academic Press.

Anderson, J. R., Boyle, C. F., Corbett, A. T., & Lewis, M. W. (1990). Cognitive modeling and intelligent tutoring. *Artificial Intelligence*, *42*, 7-49.

Anderson, J. R., Corbett, A. T., Koedinger, K., & Pelletier, R. (1995). Cognitive tutors: Lessons learned. *The Journal of the Learning Sciences*, *4*, 167-207.

Anderson, J. R., & Douglass, S. (prep). Visual attention and problem solving. Manuscript in preparation.

Anderson, J. R., & Lebiere, C. (1998). *The atomic components of thought*. Hillsdale, NJ: Lawrence Erlbaum Associates.

Andrew, S. (1991). Improving touch-screen keyboards: Design issues and a comparison with other devices. *Interacting with Computers*, *3*, 253-269.

Baldi, P., Chauvin, Y., Hunkapiller, T., & McClure, M. (1994). Hidden Markov models of biological primary sequence information. *Proceedings of the National Academy of Sciences of the United States of America*, *91*, 1059-1063.

Bharucha-Reid, A. T. (1960). *Markov processes and their applications*. New York: McGraw-Hill.

Bhaskar, R., & Simon, H. A. (1977). Problem solving in semantically rich domains: An example from engineering thermodynamics. *Cognitive Science*, *1*, 193-215.

Binstock, A., & Rex, J. (1995). *Practical algorithms for programmers*. Reading, MA: Addison-Wesley.

Byrne, M. D., Anderson, J. A., Douglass, S., & Matessa, M. (1999). Eye tracking the visual search of click-down menus. To appear in *Human Factors in Computing Systems: CHI 99 Conference Proceedings*. New York: ACM Press.

Byrne, M. D., & Anderson, J. R. (1997). Enhancing ACT-R's perceptual-motor abilities. In *Proceedings of the Nineteenth Annual Conference of the Cognitive Science Society*. Hillsdale, NJ: Lawrence Erlbaum Associates.

Cabiati, C., Pastormerlo, M., Schmid, R., & Zambarbieri, D. (1983). Computer analysis of saccadic eye movements. In R. Groner, C. Menz, D. F. Fisher, & R. A. Monty (Eds.), *Eye Movements and Psychological Functions: International Views* (pp. 19-29). Hillsdale, NJ: Lawrence Erlbaum Associates.

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

Carpenter, P. A., & Just, M. A. (1978). Eye fixations during mental rotation. In J. W. Senders, D. F. Fisher, & R. A. Monty (Eds.), *Eye Movements and the Higher Psychological Processes* (pp. 115-133). Hillsdale, NJ: Lawrence Erlbaum Associates.

Cleveland, N. R. (1997). The use of the Eyegaze system by people with severe physical disabilities. Poster presented at the Ninth European Conference on Eye Movements. System available from LC Technologies, Inc.; Fairfax, VA.

Cohen, M. E., & Ross, L. E. (1977). Saccade latency in children and adults: Effects of warning interval and target eccentricity. *Journal of Experimental Child Psychology*, *23*, 539-549.

Crawford, T. J. (1984). The modification of saccadic trajectories. In A. G. Gale & F. Johnson (Eds.), *Theoretical and Applied Aspects of Eye Movement Research* (pp. 95-102). Amsterdam: Elsevier.

DeCorte, E., Verschaffel, L., & Pauwels, A. (1990). Influence of the semantic structure of word problems on second graders' eye movements. *Journal of Educational Psychology*, *82*, 359-365.

DenBuurman, R., Boersma, T., & Gerrisen, J. F. (1981). Eye movements and the perceptual span in reading. *Reading Research Quarterly*, *16*, 227-235.

Ditchburn, R. W. (1973). *Eye movements and visual perception*. Oxford: Clarendon Press.

Ellis, S., Candrea, R., Misner, J., Craig, C. S., Lankford, C. P., & Hutchinson, T. E. (1998). Using eye tracking data to help build better web pages. Poster presented at the 42nd Annual Meeting of the Human Factors and Ergonomics Society.

Ericsson, K. A., & Simon, H. A. (1984). *Protocol analysis: Verbal reports as data*. Cambridge, MA: MIT Press.

Ericsson, K. A., & Simon, H. A. (1980). Verbal reports as data. *Psychological Review*, *87*, 215-251.

Erkelens, C. J., & Vogels, I. M. L. C. (1995). The initial direction and landing position of saccades. In J. M. Findlay, R. Walker, & R. W. Kentridge (Eds.), *Eye Movement Research: Mechanisms, Processes, and Applications* (pp. 133-144). New York: Elsevier Science Publishing.

Fischer, B. (1992). Saccadic reaction time: Implications for reading, dyslexia, and visual cognition. In K. Rayner (Ed.), *Eye Movements and Visual Cognition: Scene Perception and Reading* (pp. 31-45). New York: Springer-Verlag.

Fisher, C. (1991). Protocol analyst's workbench: Design and evaluation of computer-aided protocol analysis. Doctoral Dissertation, Department of Psychology, Carnegie Mellon University.

Fisher, C. (1987). Advancing the study of programming with computer-aided protocol analysis. In G. Olson, E. Soloway, & S. Sheppard (Eds.), *Empirical Studies of Programmers: Second Workshop*. Norwood, NJ: Ablex.

Flagg, B. N. (1978). Children and television: Effects of stimulus repetition on eye activity. In J. W. Senders, D. F. Fisher, & R. A. Monty (Eds.), *Eye Movements and the Higher Psychological Processes* (pp. 279-291). Hillsdale, NJ: Lawrence Erlbaum Associates.

Forchhammer, S., & Rissanen, J. (1996). Partially hidden Markov models. *IEEE Transactions on Information Theory*, *42*, 1253-1256.

Frederiksen, J. R., & White, B. Y. (1990). Intelligent tutors as intelligent testers. In N. Frederiksen, R. Glaser, A. Lesgold, & M. G. Shafto (Eds.), *Diagnostic Monitoring of Skill and Knowledge Acquisition* (pp. 1-25). Hillsdale, NJ: Lawrence Erlbaum Associates.

Frey, L. A., White, K. P., & Hutchinson, T. E. (1990). Eye-gaze word processing. *IEEE Transactions on Systems, Man, and Cybernetics*, *20*, 944-950.

Fuchs, A. F. (1971). The saccadic system. In P. Bach-y-Rita, C. C. Collins, & J. E. Hyde (Eds.), *The Control of Eye Movements* (pp. 343-362). New York: Academic Press.

Garlick, S., & VanLehn, K. (1987). CIRRUS: An automated protocol analysis tool (Tech. Rep. No. 6). Pittsburgh, PA: Carnegie Mellon University, Department of Psychology.

Gascon, J. (1976). Computerized protocol analysis of the behavior of children on a weight seriation task. Doctoral Dissertation, Departement de Psychologie, Universite de Montreal.

Gips, J. (1998). On building intelligence into EagleEyes. In V. O. Mittal, H. A. Yanco, J. Aronis, & R. Simpson (Eds.), *Assistive Technology and Artificial Intelligence* (pp. 50-58). Berlin: Springer-Verlag.

Gips, J., Olivieri, P., & Tecce, J. (1993). Direct control of the computer through electrodes placed around the eyes. In *Proceedings of the Fifth International Conference on Human-Computer Interaction* (pp. 630-635). Amsterdam: Elsevier.

Goldberg, J. H., & Schryver, J. C. (1995). Eye-gaze determination of user intent at the computer interface. In J. M. Findlay, R. Walker, & R. W. Kentridge (Eds.), *Eye Movement Research: Mechanisms, Processes, and Applications* (pp. 491-502). New York: Elsevier Science Publishing.

Gould, J. D. (1973). Eye movements during visual search and memory search. *Journal of Experimental Psychology*, *98*, 184-195.

Groner, R., Walder, F., & Groner, M. (1984). Looking at faces: Local and global aspects of scanpaths. In A. G. Gale & F. Johnson (Eds.), *Theoretical and Applied Aspects of Eye Movement Research* (pp. 523-533). Amsterdam: Elsevier.

Hansen, J. P. (1991). The use of eye mark recordings to support verbal retrospection in software testing. *Acta Psychologica*, *76*, 31-49.

Harris, C. M., Hainline, M., Abramov, I., Lemerise, E., & Camenzuli, C. (1988). The distribution of fixation durations in infants and naive adults. *Vision Research*, *28*, 419-432.

Hegarty, M., Mayer, R. E., & Green, C. E. (1992). Comprehension of arithmetic word problems: Evidence from students' eye fixations. *Journal of Educational Psychology*, *84*, 76-84.

Heller, D. (1983). Problems of on-line processing of EOG-data in reading. In R. Groner, C. Menz, D. F. Fisher, & R. A. Monty (Eds.), *Eye Movements and Psychological Functions: International Views* (pp. 43-52). Hillsdale, NJ: Lawrence Erlbaum Associates.

Huang, X. D., Ariki, Y., & Jack, M. A. (1990). *Hidden Markov models for speech recognition*. Edinburgh: Edinburgh University Press.

Hutchinson, T. E., White, K. P., Martin, W. N., Reichert, K. C., & Frey, L. A. (1989). Human-computer interaction using eye-gaze input. *IEEE Transactions on Systems, Man, and Cybernetics*, *19*, 1527-1534.

Intraub, H. (1981). Identification and processing of briefly glimpsed visual scenes. In D. F. Fisher, R. A. Monty, & J. W. Senders (Eds.), *Eye Movements: Cognition and Visual Perception* (pp. 181-190). Hillsdale, NJ: Lawrence Erlbaum Associates.

Jacob, R. J. K. (1995). Eye tracking in advanced interface design. In W. Barfield & T. A. Furness (Eds.), *Virtual Environments and Advanced Interface Design* (pp. 258-288). New York: Oxford University Press.

Jacob, R. J. K. (1993). What you look at is what you get: Eye movement user interfaces. *IEEE Computer*, *26*, 65-67.

Jacob, R. J. K. (1993). Eye movement-based human-computer interaction techniques: Toward non-command interfaces. In H. R. Hartson & D. Dix (Eds.), *Advances in Human-Computer Interaction* (pp. 151-190). Norwood, NJ: Ablex Publishing.

Jacobs, A. M. (1986). Eye movement control in visual search: How direct is visual span control. *Perceptual Psychophysiology*, *39*, 47-58.

Just, M. A., & Carpenter, P. A. (1992). A capacity theory of comprehension: Individual differences in working memory. *Psychological Review*, *99*, 122-149.

Just, M. A., & Carpenter, P. A. (1984). Using eye fixations to study reading comprehension. In D. E. Kieras & M. A. Just (Eds.), *New Methods in Reading Comprehension Research*. Hillsdale, NJ: Lawrence Erlbaum Associates.

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

Just, M. A., & Carpenter, P. A. (1976). Eye fixations and cognitive processes. *Cognitive Psychology*, *8*, 441-480.

Karsh, R., & Breitenbach, F. W. (1983). Looking at looking: The amorphous fixation measure. In R. Groner, C. Menz, D. F. Fisher, & R. A. Monty (Eds.), *Eye Movements and Psychological Functions: International Views* (pp. 53-64). Hillsdale, NJ: Lawrence Erlbaum Associates.

Kieras, D. E., & Meyer, D. E. (1997). A computational theory of executive cognitive processes and multiple-task performance: Part 1. Basic mechanisms. *Psychological Review*, *104*, 3-65.

Kintsch, W., & Greeno, J. G. (1985). Understanding and solving word arithmetic problems. *Psychological Review*, *92*, 109-129.

Kowler, E. (1990). The role of visual and cognitive processes in the control of eye movement. In E. Kowler (Ed.), *Eye Movements and their Role in Visual and Cognitive Processes* (pp. 1-70). New York: Elsevier Science Publishing.

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.

Kruskal, J. B. (1983). An overview of sequence comparison: Time warps, string edits, and macromolecules. *SIAM Review*, *25*, 201-234.

Laefsky, I. A., & Roemer, R. A. (1978). A real-time control system for CAI and prosthesis. *Behavioral Research Methods & Instrumentation*, *10*, 182-185.

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

Lallement, Y. (1998). A hierarchical ensemble of decision trees applied to classifying data from a psychological experiment. In *Proceedings of the Eleventh International FLAIRS Conference*. Sanibel Island, FL: AAAI Press.

Langley, P., & Ohlsson, S. (1989). Automated cognitive modeling. In *Proceedings of the AAAI-84* (pp. 193-197). Los Altos, CA: Morgan Kaufman.

Legge, G. E., Klitz, T. S., & Tjan, B. S. (1997). Mr. Chips: An ideal-observer model of reading. *Psychological Review*, *104*, 524-553.

Levy-Schoen, A. (1981). Flexible and/or rigid control of oculomotor scanning behavior. In D. F. Fisher, R. A. Monty, & J. W. Senders (Eds.), *Eye Movements: Cognition and Visual Perception* (pp. 299-314). Hillsdale, NJ: Lawrence Erlbaum Associates.

Lohse, G. L., & Johnson, E. J. (1996). A comparison of two process tracing methods for choice tasks. *Organizational Behavior and Human Decision Processes*, *68*, 28-43.

Lowerre, B., & Reddy, R. (1980). The HARPY speech understanding system. In W. Lea (Ed.), *Trends in Speech Recognition* (pp. 340-346). Englewood Cliffs, NJ: Prentice Hall.

McDowell, E. D., & Rockwell, T. H. (1978). An exploratory investigation of the stochastic nature of the drivers' eye movements and their relationship to roadway geometry. In J. W. Senders, D. F. Fisher, & R. A. Monty (Eds.), *Eye Movements and the Higher Psychological Processes* (pp. 329-345). Hillsdale, NJ: Lawrence Erlbaum Associates.

McGregor, D. K., & Stern, J. A. (1996). Time on task and blink effects on saccade duration. *Ergonomics*, *39*, 649-660.

Newell, A. (1990). *Unified theories of cognition*. Cambridge, MA: Harvard University Press.

Newell, A., & Simon, H. A. (1972). *Human problem solving*. Englewood Cliffs, NJ: Prentice Hall.

Norman, D. A. (1986). Cognitive engineering. In D. A. Norman & S. W. Draper (Eds.), *User Centered System Design* (pp. 31-61). Hillsdale, NJ: Lawrence Erlbaum Associates.

Noton, D., & Stark, L. (1971). Scanpaths in saccadic eye movements while viewing and recognizing patterns. *Vision Research*, *11*, 929-942.

O'Regan, K. (1981). The "convenient viewing position" hypothesis. In D. F. Fisher, R. A. Monty, & J. W. Senders (Eds.), *Eye Movements: Cognition and Visual Perception* (pp. 289-298). Hillsdale, NJ: Lawrence Erlbaum Associates.

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* (pp. 251-296). Hillsdale, NJ: Lawrence Erlbaum Associates.

Ohlsson, S., Ernst, A., & Rees, E. (1992). The cognitive complexity of doing and learning arithmetic. *Journal for Research in Mathematics Education*, *23*, 441-467.

Pashler, H. (1998). *Attention*. London: University College London Press.

Pavel, M. (1990). Predictive control of eye movement. In E. Kowler (Ed.), *Eye Movements and their Role in Visual and Cognitive Processes* (pp. 71-114). New York: Elsevier Science Publishing.

Pentland, A., & Liu, A. (1999). Modeling and prediction of human behavior. *Neural Computation*, *11*, 229-242.

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* (pp. 3-21). New York: Elsevier Science Publishing.

Rayner, K., McConkie, G. W., & Zola, D. (1980). Integrating information across eye movements. *Cognitive Psychology*, *12*, 206-226.

Rayner, K., & Morris, R. K. (1990). Do eye movements reflect higher order processes in reading?. In R. Groner, G. d'Ydewalle, & R. Parham (Eds.), *From Eye to Mind: Information Acquisition in Perception, Search, and Reading* (pp. 179-190). New York: Elsevier Science Publishing.

Rayner, K., Reichle, E. D., & Pollatsek, A. (1998). Eye movement control in reading: An overview and model. In G. Underwood (Ed.), *Eye Guidance in Reading and Scene Perception* (pp. 243-268). Oxford, England: Elsevier.

Reichle, E. D., Pollatsek, A., Fisher, D. L., & Rayner, K. (1998). Toward a model of eye movement control in reading. *Psychological Review*, *105*, 125-157.

Reilly, R. G., & O'Regan, J. K. (1998). Eye movement control during reading: A simulation of some word-targeting strategies. *Vision Research*, *38*, 303-317.

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. (1992). A methodology and software environment for testing process models' sequential predictions with protocols. Doctoral Dissertation, Department of Psychology, Carnegie Mellon University.

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

Russell, M. J., & Moore, R. K. (1985). Explicit modelling of state occupancy in hidden Markov models for automatic speech recognition. *Proceedings of the ICASSP*, 5-8.

Russo, J. E. (1978). Adaptation of cognitive processes to the eye movement system. In J. W. Senders, D. F. Fisher, & R. A. Monty (Eds.), *Eye Movements and the Higher Psychological Processes* (pp. 89-112). Hillsdale, NJ: Lawrence Erlbaum Associates.

Salthouse, T. A. (1986). Perceptual, cognitive, and motoric aspects of transcription typing. *Psychological Bulletin*, *99*, 303-319.

Salthouse, T. A., Ellis, C. L., Diener, D. C., & Somberg, B. L. (1981). Stimulus processing during eye fixations. *Journal of Experimental Psychology: Human Perception and Performance*, *7*, 611-623.

Salvucci, D. D. (1999). Inferring intent in eye-movement interfaces: Tracing user actions with process models. To appear in *Human Factors in Computing Systems: CHI 99 Conference Proceedings*. New York: ACM Press.

Salvucci, D. D., & Anderson, J. R. (1998). Analogy. In J. R. Anderson & C. Lebiere (Eds.), The Atomic Components of Thought (pp. 343-383). Hillsdale, NJ: Lawrence Erlbaum Associates.

Salvucci, D. D., & Anderson, J. R. (1998). Tracing eye movement protocols with cognitive process models. In *Proceedings of the Twentieth Annual Conference of the Cognitive Science Society* (pp. 923-928). Hillsdale, NJ: Lawrence Erlbaum Associates.

Salvucci, D. D., Anderson, J. R., & Douglass, S. (1997). Encoding and response strategies in complex skill acquisition. Poster presented at the Nineteenth Annual Conference of the Cognitive Science Society.

Sanderson, P. M., & Fisher, C. (1994). Exploratory sequential data analysis: Foundations. *Human-Computer Interaction*, *9*, 251-317.

Sanderson, P., Scott, J., Johnston, T., Mainzer, J., Watanabe, L., & James, J. (1994). MacSHAPA and the enterprise of exploratory sequential data analysis (ESDA). *International Journal of Human-Computer Studies*, *41*, 633-681.

Schilling, H. E. H., Rayner, K., & Chumbley, J. I. (1998). Comparing naming, lexical decision, and eye fixation times: Word frequency effects and individual differences. *Memory & Cognition*, *26*, 1270-1281.

Schumacher, W., & Korn, A. (1983). Automatic evaluation of eye or head movements for visual information selection. In R. Groner, C. Menz, D. F. Fisher, & R. A. Monty (Eds.), *Eye Movements and Psychological Functions: International Views* (pp. 31-42). Hillsdale, NJ: Lawrence Erlbaum Associates.

Sen, T., & Megaw, T. (1984). The effects of task variables and prolonged performance on saccadic eye movement parameters. In A. G. Gale & F. Johnson (Eds.), *Theoretical and Applied Aspects of Eye Movement Research* (pp. 103-111). Amsterdam: Elsevier.

Sheena, D., & Borah, J. (1981). Compensation for some second order effects to improve eye position measurements. In D. F. Fisher, R. A. Monty, & J. W. Senders (Eds.), *Eye Movements: Cognition and Visual Perception* (pp. 257-268). Hillsdale, NJ: Lawrence Erlbaum Associates.

Smith, J. B., Smith, D. K., & Kuptsas, E. (1993). Automated protocol analysis. *Human-Computer Interaction*, *8*, 101-145.

Stampe, D. M., & Reingold, E. M. (1995). Selection by looking: A novel computer interface and its application to psychological research. In J. M. Findlay, R. Walker, & R. W. Kentridge (Eds.), *Eye Movement Research: Mechanisms, Processes, and Applications* (pp. 467-478). New York: Elsevier Science Publishing.

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* (pp. 193-226). Hillsdale, NJ: Lawrence Erlbaum Associates.

Sudkamp, T. A. (1988). *Languages and machines*. New York: Addison-Wesley.

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* (pp. 455-477). New York: Elsevier Science Publishing.

Suppes, P., Cohen, M., Laddaga, R., Anliker, J., & Floyd, R. (1983). A procedural theory of eye movements in doing arithmetic. *Journal of Mathematical Psychology*, *27*, 341-369.

Tole, J. R., & Young, L. R. (1981). Digital filters for saccade and fixation detection. In D. F. Fisher, R. A. Monty, & J. W. Senders (Eds.), *Eye Movements: Cognition and Visual Perception* (pp. 247-256). Hillsdale, NJ: Lawrence Erlbaum Associates.

van Gisbergen, J. A. M., van Opstal, J., & Ottes, F. P. (1984). Parametrization of saccadic velocity profiles in man. In A. G. Gale & F. Johnson (Eds.), *Theoretical and Applied Aspects of Eye Movement Research* (pp. 87-94). Amsterdam: Elsevier.

van Hooff, J. A. R. A. M. (1982). Categories and sequences of behavior: Methods of description and analysis. In K. R. Scherer & P. Ekman (Eds.), *Handbook of Methods in Nonverbal Behavior Research* (pp. 362-439). Cambridge, England: Cambridge University Press.

Vaughan, J. (1982). Control of fixation duration in visual search and memory search: Another look. *Journal of Experimental Psychology: Human Perception and Performance*, *8*, 709-723.

Vaughan, J. (1978). Control of visual fixation duration in search. In J. W. Senders, D. F. Fisher, & R. A. Monty (Eds.), *Eye Movements and the Higher Psychological Processes* (pp. 135-142). Hillsdale, NJ: Lawrence Erlbaum Associates.

Verschaffel, L., DeCorte, E., & Pauwels, A. (1992). Solving compare problems: An eye movement test of Lewis and Mayer's consistency hypothesis. *Journal of Educational Psychology*, *84*, 85-94.

Viviani, P. (1990). Eye movements in visual search: Cognitive, perceptual, and motor control aspects. In E. Kowler (Ed.), *Eye Movements and their Role in Visual and Cognitive Processes* (pp. 353-393). New York: Elsevier Science Publishing.

Waterman, D. A., & Newell, A. (1973). PAS-II: An interactive task-free version of an automatic protocol analysis system (Tech. Rep. No. 73-34). Pittsburgh, PA: Carnegie Mellon University, Department of Computer Science.

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

Widdel, H. (1984). Operational problems in analysing eye movements. In A. G. Gale & F. Johnson (Eds.), *Theoretical and Applied Aspects of Eye Movement Research* (pp. 21-29). New York: Elsevier Science Publishing.

Winikoff, A. (1967). Eye movements as an aid to protocol analysis of problem solving. Doctoral Dissertation, Carnegie Mellon University.

Yanco, H. A. (1998). Wheelesley: A robotic wheelchair system: Indoor navigation and user interface. In V. O. Mittal, H. A. Yanco, J. Aronis, & R. Simpson (Eds.), *Assistive Technology and Artificial Intelligence* (pp. 256-268). Berlin: Springer-Verlag.

Yang, J., Stiefelhagen, R., Meier, U., & Waibel, A. (1998). Visual tracking for multimodal human computer interaction. In *Human Factors in Computing Systems: CHI 98 Conference Proceedings* (pp. 140-147). New York: ACM Press.

Yang, J., Xu, Y., & Chen, C. S. (1994). Hidden Markov model approach to skill learning and its application to telerobotics. *IEEE Transactions on Robotics & Automation*, *10*, 621-631.

Young, L. R., & Sheena, D. (1975). Survey of eye movement recording methods. *Behavioral Research Methods and Instrumentation*, *7*, 397-429.

Zingale, C. M., & Kowler, E. (1987). Planning sequences of saccades. *Vision Research*, *27*, 1327-1341.