Int. J. Human–Computer Studies (1996) **44,** 743–775

A dual-space model of iteratively deepening exploratory learning

JOHN RIEMAN AND RICHARD M. YOUNG†

MRC Applied Psychology Unit, 15 Chaucer Road, Cambridge, CB2 2EF, UK. email:richard.young@mrc.apu.cam.ac.uk

ANDREW HOWES

School of Psychology, University of Cardiff, Cardiff, Wales, UK

When users of interactive computers must work with new software without formal training, they rely on strategies for "exploratory learning". These include trial and error, asking for help from other users, and looking for information in printed and on-line documentation. This paper describes a cognitive model of exploratory learning, which covers both trial-and-error and instruction-taking activities. The model, implemented in Soar, is grounded in empirical data of subjects in a task-oriented, trial-and-error exploratory learning situation. A key empirical finding reflected in the model is the repeated scanning of a subset of the available menu items, with increased attention to items on each successive scan. This is explained in terms of dual search spaces, the external interface and the user's internal knowledge, both of which must be tentatively explored with attention to changing costs and benefits. The model implements this dual-space search by alternating between external scanning and internal comprehension, a strategy that gradually shifts its focus to a potentially productive route through an interface. Ways in which interfaces might be designed to capitalize on this behaviour are suggested. The research demonstrates how cognitive modelling can describe behaviour of the kind discussed by theories of "situated cognition". ©1996 Academic Press Limited

1. Exploratory learning

The number of commonly available software applications continues to grow, along with the number of features provided by each package. For most users, comprehensive instruction on every system they use would be impossible. Even training on the applications central to a user's job function, such as word processors and spreadsheets, may not keep pace with frequent upgrades. Nevertheless, the demands of the job and the expectations of coworkers will frequently force users to accomplish tasks using imperfectly known or newly available software.

In these situations, the user's response will be to learn exactly that part of the system needed to do the job, relying on whatever resources are available. These resources include the on-screen displays, trial and error, assistance from other users, and the information contained in manuals and on-line documentation (Rieman, in press). We use the term "exploratory learning" to describe this combination of problem solving and learning behaviour. It is a task-oriented, time-constrained

† Author for correspondence.

1071-5819/96/060743 + 33$18.00/0

process, whose primary goal is performance of the current task, with learning as a secondary aspect.

Designing effective software for today's computing environment requires a clear understanding of users' behaviour in these exploratory situations. Significant work on this front has already been done. This paper extends that research to a further level of detail, using cognitive modelling techniques informed by empirical data.

1.1. OVERVIEW OF THE PAPER

The next section of the paper discusses previous work in the area of exploratory learning. Section 3 summarizes empirical data, reported in detail elsewhere, that describe subjects' exploratory behaviour in the situation we have modelled. Section 4 is a high-level task analysis that suggests constraints on problem solving and learning with current display-based systems. In Section 5 we describe the cognitive model, IDXL. Section 6 compares the model's performance to empirical data and previous models. The paper concludes with a discussion of the model's implications for human–computer interaction (HCI) and its relationship to other work in cognitive science.

2. Prior research

Research in exploratory learning has a long history in HCI. The popularity of early computer games led several researchers to argue that user interfaces should have many of the same properties (Malone, 1982; Carroll, 1982; Schneiderman, 1983). Mastering the software should be intrinsically motivating, features should be revealed incrementally, and the system should be at least minimally useful with no formal training. Various research efforts have investigated and expanded on these ideas. In this section, we summarize research that forms the context and motivation for the work described in this paper. The discussion is organized in order of decreasing grain size, beginning with field studies and working down to cognitive models.

2.1. EXPLORATION AS A COMMON APPROACH TO LEARNING

The practical importance of exploratory learning is emphasized by a recent field study of on-the-job learning strategies of 14 interactive computer users (Rieman, 1993, in press). Informants in the study represented a wide range of background experience and occupation, including novice secretarial personnel, business financial analysts, and experienced computer scientists. Data were collected using diaries followed by in-depth interviews. The results emphasized that the overriding objective of working users is to complete their job-related duties. Given new software they may explore it briefly to gain an overview, but they will typically postpone extensive learning until it is demanded by real work. In the face of immediate demands, the users were found to rely heavily on task-oriented trial-and-error exploration, used in conjunction with printed documentation and assistance from other users.

2.2. MINIMALIST INSTRUCTION AND THE VALUE OF TASKS

The learnability of office-system interfaces in situations where novice users were given no training was the subject of studies begun by Carroll in 1981 (Carroll, Mack, Lewis, Grischkowsky & Robertson, 1985). Most users were willing and able to learn by exploration, but novices often made major and irrecoverable errors, even with the aid of basic manuals and on-line tutorials (Caroll & Mazur, 1986). This work led to the "minimalist" approach to training and manuals (Carroll, 1990). Many users have expressed a preference for this kind of guided exploratory learning.

Even with well-designed manuals, however, the breadth of coverage of discovery-oriented exploration can vary widely. A laboratory study of novice users performed by Charney, Reder and Kusbit (1990) further confirms the importance of tasks in exploratory learning. Users were given the opportunity to explore and learn a spreadsheet package under several conditions. The most effective learning was found to occur in an exploratory situation supported by a minimal-style manual, driven by a predefined set of tasks. Without these tasks, users would uncover only a limited set of the interface's features.

2.3. INSTRUCTIONLESS EXPLORATION OF PROGRAMMABLE SYSTEMS

To investigate exploratory behaviour when no manuals are available, Shrager (1985) and Shrager and Klahr (1986) gave undergraduates the unstructured task of learning to operate a "BigTrak" toy. The toy is a computer-controlled truck programmed with a simple keypad. Subjects were not instructed in the toy's controls or told what it could do. Shrager reported that subjects began their exploration with an orientation phase, then engaged in a series of problem-solving episodes, in which hypotheses about keypad functions were generated and tested. Klahr and Dunbar (1988) analysed this behaviour further in terms of a "dual search space" of hypotheses and experiments.

The computer software available to users today has many more functions than the BigTrak toy and might seem to present a more difficult environment to explore. However, any direct comparison would be misleading. As pointed out in the field studies described above, users usually investigate their software with a specific task in mind. Also, most software menus and other controls are labelled with semantically meaningful words, and each action or short sequence of actions provides immediate feedback as to its effect. In this situation, the problem for users is typically not to acquire a conceptual model of the device or to disambiguate the effect of action sequences. Rather, it is to find the control needed to achieve a well-understood result. These features of the task lead to more immediate and less reflective behaviour, as shown in the work described in this paper.

2.4. LABEL-FOLLOWING WITH SIMPLE INTERFACES

Much closer to the area of interest addressed by our work is the ongoing research of Lewis, Polson, Wharton and Rieman (1990). They have investigated task-oriented exploratory behaviour where no manuals are available, looking specifically at systems such as telephone voice-dialogue interfaces (Polson, Lewis, Wharton & Rieman, 1992) and a library database (Rieman, Davies, Hair, Esemplare, Polson &

Lewis, 1991). A key finding of the research, first reported by Engelbeck (1986), is that subjects faced with novel menus often use a *label-following* strategy, related to the identity heuristic in Lewis's (1988) EXPL theory.

Used most strictly, label-following describes a task-oriented strategy in which a user identifies controls whose labels exactly match key words in the task description, then takes the actions afforded by those controls. We use the term in a slightly broader sense, to include labels that are synonyms or category labels for a key word or concept in the task. Strict label following would lead a user to select a control labelled "Underline" for a task expressed as "Underline this work." Our looser definition would also include selections of controls labelled "Underscore" or "Formats". As such, label-following is most clearly distinguished from approaches that require system-specific knowledge or planning, such as pulling down the Macintosh's apple menu to select a printer, or using unlabelled direct-manipulation controls such as scroll bars. A label-following strategy emerges as a central feature of the model we describe later in this paper.

2.5. MODELLING WORK

Finally, a number of cognitive models have investigated exploratory behaviour with display-based interfaces. Several of these have modelled a user working with the Cricket Graph program on an Apple Macintosh, which is also the focus of the current work. The user's task in these models is to create a line graph by selecting the appropriate item from a pulldown menu and setting options in a dialogue box. (We describe the task in greater detail later in the paper.)

2.5.1. The construction–integration model

The work of Kitajima and Polson (1992, 1995) presents a model of the behaviour of expert users on the Cricket Graph task, using Kintsch's Construction-Integration (CI) theory (Kintsch, 1988; Mannes & Kintsch, 1991). The CI model's strength is the detail and theoretical motivation of its memory retrieval and lexical decision processes. The theory is a hybrid approach, combining symbolic processes for network construction with a settling-activation process for deciding among conflicting actions. Kitajima and Polson's investigation of the system's parameter space has demonstrated that it can model empirically reported error behaviour for expert users. These errors arise naturally from the model's basic memory sampling process.

A version of this model specific to novice behaviour has also been developed (Rieman, 1994). The key result of that effort was to demonstrate that the model's label following heuristic was sufficient for simple menu following, but that interactions for more complex controls, such as dialogue boxes, required a more sophisticated approach. A major shortcoming of the simple label-following approach was its failure to monitor progress on the current task. This would lead the model to repeatedly select labelled controls that matched task words, even when the information specified by those words had already been communicated to the interface. Additionally, the model describes only exploration; it has no long-term learning component. This is a limitation of the CI environment, which was developed to model relatively short-term processes of text disambiguation and comprehension.

2.5.2. The ACT-R analogy model

Rieman (1994) and Rieman, Lewis, Young and Polson (1994) describe a model of exploratory behaviour implemented in the ACT-R architecture (Anderson, 1993). The model performs the Cricket Graph task, relying on a label-following strategy that is conceptually similar to the Kitajima and Polson CI model. However, the model's behaviour reflects local decisions on individual labels, in contrast to the CI approach of considering a large set of labels simultaneously in a common network. Although the model has a representation of its task in working memory, its basic goal is always to "figure out what to do next." Its default behaviour is to look for objects on the display whose labels are the same as words in the task description, and to then consider how those objects might be acted upon to advance its task. The label-following strategy is supplemented with working memory structures that allow the model to monitor its progress and to envision the potential effects of proposed actions. The model learns from its successes. It also includes an analogical reasoning function, which can suggest an action on a novel control, based on prior experience with similar controls. A similar model of analogy in Soar (Laird, Rosenbloom & Newell, 1987; Newell, 1990) is also described in Rieman *et al.* (1994).

Although the ACT-R model considers labels one by one, it makes no detailed predictions as to the path that a user would take through a tree of menus. The exploration is random until the correct actions are identified. Rieman (1994) suggests that an effective search algorithm for display-based interfaces would be depth-first search with iterative deepening (DFID), guided by a label-following heuristic. The basic DFID algorithm is a brute-force approach that performs a depth-first search to depth one, then repeats the search going to depth two, then to depth three, etc. (Korf, 1985, 1988). The guided DFID approach (gDFID) uses the same pattern, but heuristically limits its search to items semantically related to the current task. How this strategy would be implemented in the framework of the ACT-R model is not described.

Further modelling in ACT-R is reported in Rehder, Lewis, Terwilliger, Polson and Rieman (1995). That work shows that, under certain assumptions of background knowledge concerning estimated costs and benefits of menu items, exploration may reject an appropriately labelled menu item the first time it is presented, although it would choose it on a second pass. Again, this model does not describe a detailed trajectory through the menu hierarchy.

2.5.3. TAL

The TAL (Task-Action Learning) model of Howes and Young (in press) describes a general process by which tasks and actions are decomposed and associated in a meaningful manner. A central tenet of the model is that display-based learning is a two-phase process. The first phase is recognition learning. This yields knowledge in long-term memory that a task and an action are associated; however, that knowledge will only be retrieved if both the task and the correct action are presented together a second time. The second phase is recall learning, which produces a long-term memory association that allows the action to be retrieved when the task alone is presented. The two-phase nature of learning explains a number of empirical observations about learning, including the fact that display-based interfaces are easier to learn than command-based systems, and the observation that, away from the workstation, users of display-based systems cannot reliably recall correct action sequences for systems that they can successfully use.

2.5.4. The Ayn model

Howes (1994) uses a theoretical approach related to TAL to define the Ayn model of exploratory learning. The model reflects three broad aspects of behaviour with interactive computer systems: (1) Users learn devices by exploration; (2) their behaviour becomes faster and more error free with practice; and (3) much of their knowledge is display-based, i.e. it is dependent on cues provided by the display (Howes & Young, in press). The Ayn model describes search through a menu structure similar to the Macintosh style menus used in the Cricket Graph task. The structure is somewhat simplified in that more complex controls, such as dialogue boxes, are represented by another level of simple menus. In Ayn, as in Rieman's (1994) ACT-R model, behaviour arises out of local decisions, without explicit consideration of the global search space.

Ayn's local decision-making algorithm relies on four sources of knowledge. Semantic match knowledge allows irrelevant labels to be avoided. Failure detection knowledge identifies a dead-end in the search and describes how to cancel out of a menu, thus returning to the top level. Recognition knowledge, learned as Ayn explores, flags the fact that a given menu item has been tried before. Finally, task-control knowledge, which is learned more slowly than recognition knowledge, indicates that a higher-level menu item leads to success or to certain failure. Each of these forms of knowledge, and the processes through which they are acquired, reflect limits on the cognitive capabilities of the user (Howes, 1994; Howes & Young, in press).

Ayn makes detailed predictions about the patterns of behaviour that will be observed as the user performs a task-oriented search of a menu hierarchy. Those patterns change with experience. Early runs of the model reflect novice exploration, including a distinctive search pattern that initially picks an attractive item on the top menu, investigates a single path to the point of failure under it, and then shifts to a different attractive top item for another single-path foray. Later runs show intermediate skill that performs the task with occasional errors. Eventually the model exhibits expert, error-free performance.

2.6. SUMMARY OF RELEVANT PRIOR WORK

The research described in the preceding paragraphs has demonstrated that exploratory learning is a frequent, effective, and often preferred method for a user to learn about an interactive system. Most often, the learning is task-driven and incremental: users who are generally familiar with their software are faced with a job that requires them to extend their knowledge slightly to cover a previously unused feature. In such a situation, label following can be an effective approach, but it needs to be supplemented with strategies for guiding search through an interface and keeping track of what has been tried and learned. Prior work also predicts menu search patterns that may arise, given a potentially large hierarchy of menus and limited cognitive resources for retaining information as it is revealed.

3. Empirical observations

The models of Cricket Graph and the work in Ayn make various predictions about users' exploratory behaviour, at various grain sizes. In this section we examine similar categories of behaviour in empirical reports of subjects performing the Cricket Graph task. Details of the studies and results are reported in Franzke and Rieman (1993), Franzke (1994, 1995) and Rieman (1994). The first two studies analyse the behaviour of a large number of subjects. We summarize the relevant parts of the studies here.

3.1. EXPERIMENTAL TASK AND PROCEDURE

The studies examined users as they explored the use of the Cricket Graph† program on an Apple Macintosh. All exploration was task-oriented: subjects were given a defined result to achieve. Subjects had experience with the Macintosh and knew its interface conventions, but had never used nor received training on the Cricket Graph program.

Cricket Graph is an application for creating graphs (line, bar, pie, ec.) from numerical data represented in a spreadsheet format. Figure 1 shows the Cricket Graph display. The experimental task given to subjects had two parts: (a) starting the Cricket Graph program and creating a new graph from a supplied data file, and (b) editing the graph to match its format to a sample graph. The sample differed from the default created by Cricket Graph in a number of ways, such as the style of the graph title and axis labels, the wording of the Y-axis label, and the symbols used

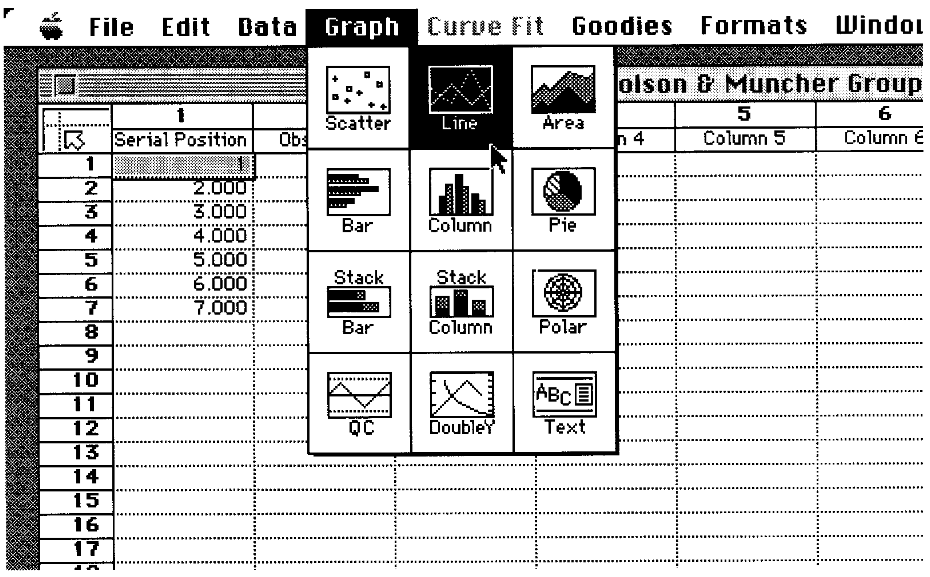

FIGURE 1. The Cricket Graph menu interface.

† Cricket Graph™ is produced by Computer Associates, Inc., One Computer Associates Plaza, Islandia, NY 11788, USA. This paper describes version 1.3.2, designed by J. Rafferty and R. Norling, and released in 1989.

for the data points. In the studies reported in Rieman (1994), subjects had to decide on the types and the order of modifications they wanted to attempt. The experiments by Franzke (1994, 1995) controlled the order of modification.

In all cases, the subject's instructions were presented on screen in a HyperCard stack, which included general instructions, a sample graph, and details about the data, graph type, and data-to-axis mappings. Subjects were not allowed to use the Cricket Graph manual or on-line help, nor to ask the experimenter for assistance. If subjects persistently failed to discover an action and continued to explore the same parts of the interface, the experimenter would provide a hint to help them continue. The users' on-screen actions and thinking-aloud protocols were video taped.

3.2. SUMMARY DESCRIPTIONS OF EXPLORATORY BEHAVIOUR

Franzke (1994, 1995) analysed the protocols of 76 subjects. All subjects completed the Cricket Graph task by exploration, with occasional hints from the experimenter. The analysis identified three key factors that influenced the time needed to discover an action.

The first factor was the quality of the label associated with the correct control. Franzke distinguishes four levels of semantic distance to describe labels: (1) words identical to a key word of the task as it was presented in the instructions, such as "graph"; (2) words that were synonyms to a word in the task, such as "chart"; (3) words that were semantically related but required some inference, such as "drawing tools"; and (4) words that had no clear connection to the task, such as "file". The analysis showed that the time to discover actions increased monotonically along this scale. Actions with the best labels were discovered almost immediately, with a mean time of less than 15 s. The mean time needed to discover the least meaningful labels was almost 2 min, indicating that many subjects required assistance from the experimenter.

The second factor influencing exploration times was the number of objects on the display. Discovering an action in a display with ten or fewer objects was relatively rapid, but exploration became more time consuming for more complex screens. Further, there was an interaction between label quality and display. A good label might be discovered rapidly even in a complex display.

The third factor identified in the analysis was the type of interaction associated with the label. Dialogue box controls and menu items were discovered most quickly, while direct manipulation actions took longer. Direct manipulation was especially difficult to discover in the editing phase of the Cricket Graph task, where subjects had created a graph using menus and then had to discover that the objects in the graph could be double-clicked or dragged.

3.3. DETAILED ANALYSES OF EXPLORATORY BEHAVIOUR

Franzke's analysis identified key factors influencing the ease with which interactive controls could be discovered and learned, without examining the actual process of exploration. The analysis in Rieman (1994) complements that research by investigating the low-level exploratory behaviour of 20 subjects. Two categories of behaviour are of particular interest here: label following and menu search.

3.3.1. Label following

Again, "label following" describes a strategy in which a user identifies a label that matches a key word or concept in the task, then takes the action that the associated control affords. One point within the Cricket Graph task where label-following would clearly succeed is the situation where Cricket Graph has been started, the data file has been loaded, and the next step is to create the initial, default graph. The goal should be to "create a graph", and a menu bar item labelled "Graph" is the correct control to select. Franzke's analysis shows that subjects will quickly select the Graph menu, but does not describe the behaviour leading up to that selection. The protocols were examined by Rieman (1994) with this question in mind.

A full 90% of the subjects located the correct menu item. However, only 15% of the subjects moved the mouse cursor directly to the Graph menu and pulled it down. Another 50% found the Graph menu without problem, but only after moving the mouse cursor over other parts of the screen. In half of these cases, subjects pulled down and examined the Graph menu at some point, but then moved on to examine other menus before returning to Graph. Another 25% of subjects repeated their previous action once or more (starting the program), then went on to scan the menus and find Graph.

These protocols show that label-following may be an accurate high-level description of users' strategies at this point, but it fails to reflect the low-level mixture of visual scanning and active menu exploration that precedes the actual selection of a control.

3.3.2. Menu search patterns

To gain further understanding of users' exploratory behaviour, we examined a situation in which the match between task and label was not so obvious. In the Cricket Graph studies, as soon as the default graph was created, subjects were asked to modify it to match the sample graph shown in the instructions. The modifications included changing the text of one axis label and the graph title, changing the typographic style of several items, and changing the shape of the data points. Each of these changes can be accomplished only by double-clicking on the screen object to be modified. However, subjects in the experiment were experienced Microsoft Word users who were accustomed to using menus for formatting. In keeping with this experience, subjects invariably began searching the menus for an appropriate formatting option. Because their search was doomed to fail, this point in the interaction was ideal for observing extended menu search strategies.

The theoretical analysis in Rieman (1994) had suggested a form of depth-first search with iterative deepening, limited to items with semantically meaningful labels (gDFID). With the Macintosh interface, this would be realized by first scanning the menu bar, then pulling down all attractive menus and viewing their options, then trying all attractive menu items to observe their dialogue boxes or effects, then delving progressively further into potentially useful dialogue boxes. The Ayn algorithm, in contrast, would pick a single attractive menu bar item, immediately try a single attractive option in the pulldown menu, and immediately OK or cancel a dialogue-box configuration. Ayn would then investigate a different selection from the top menu bar.

Mouse-cursor moves, menu selections, and verbal protocols of all 20 subjects were examined for search patterns that might arise from a gDFID strategy, and a subset was subsequently examined for patterns consistent with Ayn. Neither strategy could be clearly confirmed or rejected. Only one of the subjects obviously scanned each item in the menu bar before first pulling down the menus, which sDFID is not a common approach. And once the menus had been pulled down and examined, the next level of search was almost never pursued before scanning one or more of the pulldown menus again, which suggests that the Ayn approach does not tell the whole story. However, we do not know which menu labels subjects identified as semantic matches to their task, if any, and neither gDFID nor Ayn describes behaviour where there are no semantic matches.

If this was indeed a situation with no semantic matches, then we might expect users to initiate a more exhaustive search. However, instead of expanding their search by trying additional items from the pulldown menus, many subjects would repeatedly scan the pull down menus or a subset of them with increasingly greater attention to each item. This *iteratively deepening attention* was shown by several forms of observable behaviour: (1) spending more time looking at a pulldown menu on a later pass, (2) sliding the mouse cursor down a menu that had only been visually scanned on an earlier pass, and occasionally (3) reading the menu items out loud on a later pass. Iteratively deepening attention, as evidenced by one or more of the three forms of behaviour just listed, was identified in 75% of the protocols examined in Rieman (1994). Periodically, the attention phase would lead to selection of one of the menu items. The obvious interpretation of this behaviour, which is an important feature of the model described later in this paper, is that subjects were putting greater effort into thinking about the meaning of each item on subsequent passes.

A second observed pattern of behaviour was a spatial search strategy within menus: 65% of the protocols showed full or partial left-to-right or right-to-left scans as subjects pulled down menus from the top menu bar. In a partial scan, users might skip the leftmost Apple, File, and Edit menus, a strategy that could be called "label avoidance". Most users also worked from top to bottom of pulldown menus as they considered or tried items within them. None of this behaviour is surprising, but it shows again that label-following is not a complete story.

4. Task analysis

The empirical data show behaviour that existing models do not predict. Users will go beyond strict label-following and select synonyms or items with an inferred relationship to their task, although the need to do this slows their progress. They are much more willing to try actions at the level of the top menu bar than in the pulldown menus. And the order of their search is affected by the layout of the screen. Understanding the reasons for this behaviour could be productive in designing better interfaces or training users in exploratory strategies.

To lay the foundation for a model that better matches the data, we first reconsider our analysis of the exploratory-learning task. The gDFID algorithm and Ayn characterize exploration of the menu tree as trial-and-error investigation of a search space. As noted in Howes (1994), this is an abstraction that simplifies many of the characteristics of the actual interface. It may be that one or more of these simplified characteristics needs to be treated in greater detail by the model.

4.1. SIMPLIFYING ASSUMPTIONS REVISITED

One simplification is that each menu label is assumed to be either semantically relevant or not relevant to the task, on a binary scale. Labels that are relevant identify actions that should be attempted, while all other labels identify actions that will never be attempted. This simplification cannot be maintained if we are to model Franzke's four levels of semantic match.

A second simplification is that a top-level menu bar and a pulldown menu are treated as two instances of the same kind of conceptual object. They both offer a selection set of possible actions, which lead to success, failure, or an unknown state. But the data show that users are more reluctant to choose items from pulldown menus than from the top menu bar. Some factor related to this difference needs to be included in our model.

Expanding on the analysis in Rehder *et al.* (1995), we propose that the missing factor is the potential cost, measured primarily in time, of taking the wrong action in the interface. The cost of pulling down the wrong menu from the top menu bar is negligible. It can be undone immediately by releasing the mouse button. But the cost of selecting the wrong item from a pulldown menu is unknown. It may produce a dialogue box that can be cancelled, but it might also restructure the entire spreadsheet or send a file of garbage to the printer. The user who has worked with other applications in the current environment (as is typically the case with exploratory learning) will be aware of this distinction, even though the precise effect of trying an individual menu item cannot be predicted.

Cost is also a consideration in deciding which of several items to examine next within a menu. For certain kinds of interface object, such as the top menu bar on a dialogue box, exploring in an order supported by the screen layout (e.g. scanning the menu bar left-to-right, instead of at random) reduces the cost associated with physically moving the mouse cursor from one item to the next. For these and other controls, a physically linear search through a list of items may also make it cognitively easier to keep track of which items have been tried. For example, the user can simply remember that "everything to the left of the Graph menu" has been examined, instead of having to remember the individual items.

4.2. THE COST OF KNOWLEDGE

The potential cost of wrong actions encourages the user to apply any available background knowledge to help select the correct item from a pulldown menu. We assume that the relevant knowledge about an item is not simply limited to its identity with a task word, or even with a synonym of the word. For example, at first glance a label may be an exact match to the user's task; but a more extensive probing of memory might retrieve past cases in other applications where the same label was used for a very different purpose. Retrieving and interpreting additional knowledge may require the user to apply deliberate recall strategies, consider synonyms or related terms, and even form analogies to experiences with other software. These processes also have a cost, again measured in time, and the cost of some cognitive processes will be higher than others.

A further complicating factor is that the knowledge of a menu item may be improved by further exploration in the interface. As modelled in Ayn, exploration may teach the user that an attractively labelled menu does not lead to any useful states. More sophisticated learning can occur as well. A user may initially believe the Font menu contains both Helvetica and Times. But if she finds Helvetica under the Style menu, she could infer that Times is also there, without further search.

4.3. RATIONAL DUAL-SPACE SEARCH

The information needed to accomplish the user's current task, therefore, is distributed across two different spaces: the external space of interface states and the internal space of knowledge. The spaces are intimately linked. Each menu item is the root of a tree of interface actions, but it is also the source of a network of semantic relationships. Both external and internal spaces can be probed to varying depths at varying costs. And the knowledge about each menu label can be incrementally redefined by the knowledge acquired as the external interface is explored, subject to the user's ability to identify appropriate associations and retain them.

In such a situation, the rational user must strike a balance between external and internal exploration, considering the predicted costs and benefits of each. We use "rational" here in very much the same sense as Anderson (1990; 1993): rational behaviour chooses the course of action most likely to succeed at the least cost, including computational costs. However, our argument is qualitative and symbolic, without the quantitative features that characterize Anderson's work.

In exploring a new piece of software, the rational calculations of benefit versus cost must be estimated based on past experience with other applications in the same environment. Experience with WIMP (windows, icons, menus, and pointers) systems in general will show that pulling down menus and scanning them is a cheap and valuable activity. Rationally deciding to select an action from those menus, however, is much more difficult. It requires the user to balance the high cost of possible failure against the high value of possible success. This decision, in turn, must be balanced against the potential cost and benefit of further search in the internal knowledge space, which might retrieve additional information about a label and yield a more reliable prediction of the associated action's effect.

The empirical protocols show how experience has shaped the behaviour of the subjects tested. In particular, the protocols confirm that the perceived benefit-to-cost ratio of pulling down menus on a Macintosh is very high. Users do this almost without hesitation. However, the benefit-to-cost ratio of actually selecting from a pulldown menu, without first understanding what the selection might do, is apparently quite low. Only very good label matches, therefore, will be rapidly selected, while approximate matches must be focused on and thought about before actually being selected.

The empirical evidence also suggests that users are sensitive to the varying costs involved in retrieving and interpreting existing knowledge. Incrementally deepening attention allows the user to postpone in-depth consideration of marginal labels until other candidates have been scanned and briefly evaluated. In other words, users put a small amount of time into thinking about each label, in an attempt to identify the correct action with minimal cognitive effort. If that fails, they scan promising parts of the interface again and apply more costly cognitive strategies. The overall picture that emerges is of a dual-space search that gradually expands outward in both the external space of menu selections and the internal space of considered implications of those selections.†

5. The IDXL model

The model we describe in this section, IDXL, has been developed to reflect the empirical data and theoretical arguments just presented. It also draws heavily on previous modelling work, and integrates several ideas and exploratory strategies into a coherent single approach. As noted earlier, our modelling work in this domain has its antecedents in work done in ACT-R, Soar, and the Construction-Integration theory. The IDXL model is in Soar. As we will explain below, the learning mechanisms in Soar provide important support for iteratively deepening attention. Additionally, constraints on external interaction in Soar are an incentive to make the model highly interruptible, which aligns well with the need for rapid shifts between external and internal search.

One key point needs to be emphasized at the outset: IDXL describes cognitive and physical behaviour at a finer level of detail than the other models we have discussed. Specifically, the IDXL model describes shifts in visual attention as the interface is scanned. The model incorporates a visual buffer, the contents of which are limited to a single on-screen item. Global decision processes such as "pick the best menu item on the screen" must be implemented using this single-item buffer, combined with working memory and long-term memory of items previously viewed.

5.1. SOAR: STATES, OPERATORS, AND CHUNKS

The IDXL model is implemented in Soar (Laird *et al.,* 1987; Newell, 1990). We present a brief description of the relevant parts of Soar in this section, as a foundation for later discussion of key features of the model. The model is defined principally in terms of operators and states. In the Soar architecture, the *state* is a working memory construct that contains all the declarative facts known about the current situation. *Operators* are cognitive actions that are proposed in response to the current state. If multiple operators are proposed, *preference* knowledge is applied to select the best. The selected operator is then applied, which will change working memory. For example, in response to a state that expressed the task goal of "find the sum of six tens", the add and multiply operators might be proposed. Preference knowledge might indicate that multiply is preferable. The multiply

† To avoid confusion, we point out that the two spaces we describe here are different from those used in the analysis of Klahr and Dunbar (1988), who propose a space of hypotheses and a space of experiments. They are also different from the two spaces proposed by Payne, Squibb, and Howes (1990), whose analysis refers to a task space and a device space.

operator would be selected and applied, placing "sixty" into working memory. This, in turn, would create a changed state in which different operators could be proposed, selected and applied.

If an operator is chosen but it is not immediately obvious how to apply that operator, then the system has reached an *impasse.* Impasses can also occur for other reasons, such as the lack of preference knowledge to decide among proposed operators. In any impasse situation, the Soar architecture will automatically create a sub-goal. This new goal will have its own state, "under" the current state. In this state the system will attempt to bring existing knowledge to bear in order to resolve the impasse. For example, the system might not immediately know the answer to "multiply six times ten", but it might have knowledge of an algorithm that would supply the answer. The multiply operator would impasse and the algorithm would be applied in the sub-goal state. When the answer "sixty" was returned to the state above, a Soar *chunk* would be formed, encoding the result permanently in long-term memory. If the need to multiply six times ten occurred again under similar circumstances, the chunk would fire, avoiding the need to sub-goal and work through the algorithm again.

In a Soar model of interactive behaviour, certain operators may change working memory in a way that causes the model to take (simulated) physical actions. For example, an operator might put "press mouse button" into the part of memory that controls the hand. Conversely, perceptual input from the (simulated) external world may add information to working memory, causing new operators to be proposed. The constraints of Soar allow motor output and perceptual input to occur only through the top state. There are fundamental reasons for this, primarily having to do with considerations of chunking, such as the consistency of behaviour before and after learning (Laird & Rosenbloom, 1995). Newell points out that after sufficient learning, highly practised behaviour takes place (at least for short stretches of time) in the context of the top state, using chunks learned from earlier processing in sub-goals. If those sub-goals had used their own local state for input and output, then for skilled behaviour—when there are no impasses and no sub-goals—it would be impossible for perception and action to occur (Newell, 1990: p. 256).

All long-term knowledge in Soar is defined as productions, and chunks are simply productions that the system itself creates. Theoretically all the knowledge in a Soar model could be acquired through chunking. As will be described in detail below, an important characteristic of the IDXL model is the extent to which it uses chunks to incrementally build its knowledge of the interface being explored. This requires that certain operators frequently impasse and sub-goal, and that the sub-goals be processed rapidly to form simple but numerous chunks.

5.2. DETAILS OF THE IDXL MODEL

The implemented model addresses the first part of the task used in the empirical testing: it starts the Cricket Graph program, selects the Line option under the Graph menu, and clicks the OK button in a dialogue box. The code has two fundamental parts: a minimal simulation of the Cricket Graph interface with which the model interacts, and the IDXL model itself. Parts of the task have been simplified to focus on issues of interest: the model starts the Cricket Graph program but does not have to load the data file, and the line-graph dialogue box includes nothing except the OK and Cancel buttons.

5.2.1. *The Cricket Graph simulation*

The simulation presents the model with a symbolic representation of the display object that the model is visually attending to, and in a form that approximates the result of perceptual processing. When the model directs its visual attention to the Graph menu item, for example, the simulation reports that this is a menu-bar item, with the label Graph, at a specific location. It also flags the presence of the mouse cursor if it is on that menu item.

The simulation also responds to physical actions initiated by the model. If the model chooses to press the mouse button while the cursor is on the Graph menu, then the simulation changes its state to reflect that the pulldown menu has dropped, and it sends a "visual alert" to the model's visual buffer, simulating the abrupt visual change that would draw attention to the newly visible menu. It is then the job of the IDXL model to move the visual focus to that menu and to scan and interpret its contents.

5.2.2. *What the model knows about the task*

At the outset, the model is supplied with a task description in working memory. The description is multipart and hierarchical: the task is to start Cricket Graph and create a line graph. It is represented in subject–verb–object format, with modifiers (Figure 2). In addition to this information in working memory, the model has knowledge of Macintosh menu conventions expressed as productions in long-term memory. Specifically, it knows that pressing the mouse button while the cursor is on a menu bar item is an appropriate action. Its preference knowledge suggests performing that action immediately, reflecting experience that the action is relatively safe (low potential cost). It also knows that releasing on an item in a pulldown menu is appropriate, but here its preferences call for further consideration before action, since the cost of undoing such an action may be high.

5.2.3. *Scanning as a default*

The default behaviour of the model is to *scan* the interface. Scan operators, based on work reported by May, Scott and Barnard (1995), move the visual focus left, right, up, or down, or jump to another part of the screen, or "zoom" in or out to focus attention on a smaller or larger part of the visual field (e.g. the menu bar, or the File item in the menu bar). Additional preference knowledge, such as "continue scanning in the same direction if possible", helps control the scan and achieve good coverage of the screen.

```
Task:
  Schema:
    Subject: I, Verb: start, Object: Cricket-Graph
  Schema:
    Subject: I, Verb: create, Object-modifier: line, Object: graph
```

FIGURE 2. Task representation.

Scan operators are always options that the model will consider, under any circumstances. Even if it has no task description, the model will scan. Thus, scanning is not driven by the model's task goals. In fact, it is an activity that could acquire new tasks or, if a task is already specified, help identify the sub-tasks that lead to a solution. (The current system does not model these functions.)

In the Soar implementation, the knowledge of which scan operators to propose, how to select among those operators, and how to apply those operators is encoded as productions that apply in the top-state. This models a user who is familiar with the interface and has well established visual search patterns. The knowledge includes items such as a preference for horizontal over vertical scans, a preference for left-to-right over right-to-left, and an overriding preference to continue a scan in the current direction whenever possible.

5.2.4. Incremental comprehension

The model's second main operation is to *comprehend* what it has just scanned. Whenever a new item appears in the visual buffer, the model attempts to comprehend that item in the context of the current task. Comprehension may be as simple as a judgement that this item has no relationship to the task. Or, the comprehension operator may note that the scanned item is a label that matches some key word in the task.

A key feature of the model is that comprehension is performed incrementally, allowing frequent interruptions to acquire or respond to new information from the external world. Comprehension is implemented in the model as a Soar operator, which impasses and produces a sub-goal whenever it tries to comprehend an item. Within the sub-goal state, the model applies appropriate knowledge and procedures and returns comprehension information as soon as possible. As soon as any information is returned, the sub-goal is terminated, and the model returns to its top state. At this point, further scanning can take place. Alternatively, if the initial attempt at comprehension returns information suggesting that the comprehended item might be useful, the model can delay further scanning and attempt to comprehend the item again in greater depth. Figure 3 shows the kinds of knowledge that successive applications of the comprehend operator can return.

Comprehend	→	exact match or no-match of label to task word
Comprehend	→	synonym or no-synonym of label to task word
Comprehend	→	afforded action (if known), or suggestion to try recall
Comprehend	→	suggestion to try analogy, if recall failed
Comprehend	→	suggestion to ask for instruction, if analogy failed
Comprehend	→	recommended action, if recall or instruction or analogy has identified one
Comprehend	→	confirmation that it is safe to try the recommended action, after thinking ahead

FIGURE 3. Information that may be returned by successive applications of the Comprehend operator. Least costly (time-consuming) comprehension operations are performed first. More difficult operations are performed on successive applications, where appropriate. Not all operations may be appropriate for a given object.

Incremental comprehension reflects the cost considerations identified earlier in the task analysis. The cost of fully comprehending any single item is high, and the item's label is a poor predictor of the item's relevance to the task. Incremental comprehension allows the user to gradually develop a better understanding of one or more items, postponing more costly cognitive operations until their need is clear. The search for direct label matches on the first scan, for example, is a very quick (low-cost) operation. On a subsequent scan, the search for synonym matches requires the more time-consuming operation of probing the semantic relationship between the label and the task words. Additional and increasingly time-consuming comprehension processes include thinking ahead to predict the possibilities of success for a control, as well as using analogical reasoning to suggest the affordance associated with a new type of control.

Incremental comprehension also allows the search process to be easily interrupted after each quantum of comprehension is completed. Interruption may be required to gather more information about a candidate item, or to shift consideration to another candidate, or to respond rapidly to unexpected input from the external world.

Each time the model incrementally comprehends an item and returns information to its top state, the Soar architecture forms a chunk, a record in long-term memory associating the comprehended item with its result. Subsequent attempts at comprehending the same item can use the chunk to build on the first result without repeating the work. This will be true not only if the subsequent attempts at comprehension follow immediately, but also if they follow after further scanning. For example, given the task of "create a Plot", the model might scan the menu item labelled "Graph" and quickly return a "no match" result. On a subsequent pass over the menus, the model could note the previous result and immediately try the more effortful comprehension process of identifying synonyms.

5.2.5. Scanning, comprehending, and acting

As just described, whenever the comprehension operator returns information suggesting that an item on the display might be useful for the current task, the model has a choice. If the item seems only marginally likely to be useful, because its label is only a synonym to a task word, the model may decide to scan further. If the item seems highly likely to be useful, because its label is an exact match to a task word, the model may try to comprehend the item's meaning further, typically by considering what action the object affords. Further comprehension can suggest the likely result of that action and consider whether it is related to the task and safe to try. Eventually, the comprehension process will place information on the top state that causes physical action operators to be proposed, such as moving the mouse or releasing on a pulldown menu item.

At some point, the model's preference knowledge will indicate that the proposed physical action is a better choice for the next operator than further scanning or further comprehension, and the action will be performed. For safe actions and direct matches, this could happen very early: the first time the model sees the Graph menu, it may move the mouse cursor there, because moving the mouse has low cost and is always reversible. For less likely actions or those with greater potential cost, such as releasing on a pulldown menu item whose effect is uncertain, the action may be preceded by an extended mixture of scanning, comprehending, scanning other menu items, and returning to the target item before any action is performed.

The decision making of the current model is entirely local to the item currently in visual focus. No list of potential candidates is maintained; instead, the model relies on repeated scanning to bring earlier candidates back into consideration, at which point earlier chunked knowledge will apply and further comprehension can take place.

The exact search path the model takes depends on the quality of the matches and the preference knowledge, which a real user would have acquired through experience with other systems. A user who had often worked with a well labelled, friendly system, with good undo facilities, would have knowledge that preferred actions after only two or three applications of the comprehension operator. A user with experience in systems where undo was poorly implemented or labels were often misleading might have knowledge that preferred further scanning or comprehension in the same situation. (Section 6.1 gives additional details on the effect of preference knowledge.)

5.2.6. More costly strategies: instruction, recall, and analogy

In a real-world situation, users will not rely solely on label following and trial and error to learn how to use their software. They will use whatever resources they have available, especially manuals, advice from other users, and on-line help (Rieman, in press). To demonstrate how those strategies are integrated into the users' behaviour, the IDXL model has the ability to ask for instructions, to recall previous instruction, and to perform analogical reasoning that extends instructed knowledge to cover similar situations.

The instruction-taking facility of IDXL learns using the two-stage process reported by Howes and Young (in press). Instruction learning poses a difficulty for Soar. If a Soar model is implemented that simply asks for and follows instructions whenever it reaches an impasse in its current task, then it will learn to do exactly that: ask for and follow instructions. The instructions must always be given for the learned ability to be useful. Of course, the aim of presenting instructions is to have the model learn to do the task *without* those instructions. [This difficulty is a version of what Newell (1990), describes as the "data-chunking problem."] The solution involves a model that must be presented with the same task twice, allowing learning to be performed in two stages. In the first stage, the model learns to *recognize* the instructions. In the second stage, it tries to generate ("guess") the instructions for itself, using the recognition knowledge from the first stage to know when it has generated them correctly. That second stage results in *recall* knowledge for the instructions, which can be used directly if the task occurs again.

In a little more detail, the process runs as follows. When the model decides it needs instructions, it poses a question, formed in terms of its task. If the task is represented in subject–verb–object format as, "I start cricket-graph" then the model will ask, "how do I start cricket-graph"? The simulation returns an answer, "you double-click cricket-graph", which the model simplistically parses. The IDXL model includes no real natural language capability, but the parsing step is important, because it involves impasse-driven behaviour that forms *recognition chunks.* The next time the program is asked to start Cricket Graph, it will not immediately be able to recall the procedure.

Instead, it will use the task description and the on-screen display to generate potential answers, identical in form to what it would receive from instruction. For example, it will generate, "you drag-to-trash cricket-graph". After several incorrect instructions, it will generate, "you double-click cricket graph". As soon as this imaginary answer is generated, the recognition chunk formed by the earlier parsing will fire, indicating to the model that the answer was received earlier as an instruction. At this point, the model will learn a *recall chunk,* which directly associates the answer with the question. The third time the task is posed, the recall chunk will fire immediately, without the need to generate and test possible answers. (The generate-and-test performed in the second phase actually occurs the first time the task is presented as well; the model only asks for instruction after all generated candidates fail.)

To extend its knowledge beyond instruction, IDXL can use analogy. The analogy process is virtually identical to that described for the Soar model reported in Rieman *et al.* (1994). If the model identifies an object whose label indicates that it might advance the task, but the model does not know what action the object affords, then it can try to predict the action by analogy to past cases. For example, assume the model knows how to start Microsoft Word but does not know the general rule that double-clicking a program icon starts the program. Given the task of starting Cricket Graph, the model can perform the following analogical reasoning: Word and Cricket graph are both programs; double-clicking the Word icon starts Word; the task is to start Cricket Graph; by simple substitution, double-clicking the Cricket Graph icon might be the way to start Cricket Graph.

An important feature of the IDXL model is that instruction taking and analogy are related processes. The recall chunks learned through instruction can be used by analogy. If recall chunks have not yet been formed, the model will use the generate-and-test process to try to form them on the basis of recognition chunks. If analogy fails, the model will ask for instruction. If analogy succeeds, the chunks produced will be identical to the recognition chunks learned through initial instruction, and the task will need to be presented again to allow the generate-and-test phase to produce recall chunks. This use of chunks formed for a specific purpose to support an entirely different later goal is an approach also found in Huffman (1993, 1994) and in Vera, Lewis and Lerch (1993).

As with all exploratory strategies, the preference knowledge that helps choose among the strategies of recall, asking for instruction, and analogy is dependent on the user's background experience, reflecting the costs and potential benefits of each strategy. Analogy, for example, requires several cognitive operators (recalling a past case, identifying a mapping, performing a substitution), and the action suggested by the process is not guaranteed to succeed. As such, analogy has a poor benefit-to-cost ratio, and the model prefers to use recall first. These preferences can be easily modified in the model by adding or deleting simple productions.

5.2.7. Shallow goal stacks and sidegoaling

Incremental comprehension helps to make the model highly interruptible. However, although early comprehension operations involve only simple operators such as label matching, later operations may require a significant amount of internal processing. Analogy is an example of such a process. It requires that similar prior cases be identified, similarities and differences compared, a mapping performed, and a result extracted (Holland, Holyoak, Nisbett & Thagard, 1986; Rieman *et al.* 1994). In a traditional sub-goaling architecture, this additional work would be implemented as a deep stack of sub-goals under the comprehension state [Figure 4(a)]. When the processing in the sub-goals is complete, the final answer, double-click the Cricket

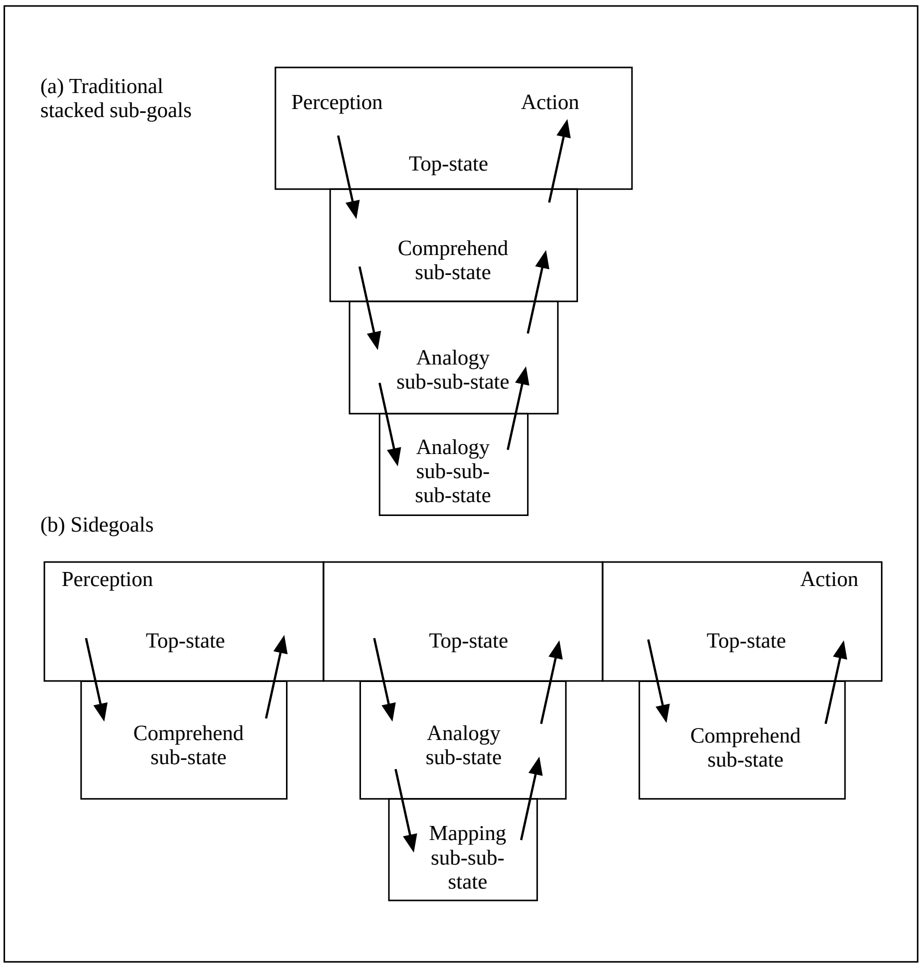

FIGURE 4. Traditional sub-goals compared to sidegoals. In traditional sub-goaling, the architecture maintains the previous goal and context, which can be resumed automatically when a sub-goal is accomplished. In sidegoaling, the previous goal is replaced by the new goal. Return to a previous goal occurs only if the internal context and the perceived external environment cause that goal to arise again after the sub-goal is completed.

Graph icon, would be returned to the comprehension state for immediate use. The deep stack of sub-goals in this approach includes a considerable amount of temporary data about the developing solution, and is inherently difficult to interrupt.

In the IDXL model, by contrast, when the need for analogy is identified in the comprehension state, that state proposes analogy as an operator in the top state—the same state where scanning and physical actions occur [Figure 4(b)]. The comprehension operator is then terminated, and analogy proceeds. When analogy returns its result, it returns it to the top state. Comprehension will now be proposed again (because the last attempt at comprehension was terminated before a result was returned). This time, the information produced by analogy will be available, so analogy will not be proposed again.

We refer to this process as *sidegoaling.* A distinguishing feature of sidegoaling is that neither the architecture nor working memory retains an explicit description of the state to which the model should return after the sidegoal is processed. Instead, the context information that led to the original state (comprehension, in this example) causes it to occur again after the sidegoal is completed. It is important to note, however, that if the context information were to change while performing the sidegoal—for example, if a colleague interrupted with a hint, or urgent email arrived—then the model might decide to act on that information instead of returning to its original process.

5.2.8. Sidegoaling for external interaction

As pointed out earlier, external interaction in the Soar architecture must be performed from the top state. This leads to other situations where comprehension processes need to be interruptible. In using analogy to suggest the afforded action for the Cricket Graph icon, for example, the model tries to recall other programs that it has used. To cue its memory, it decides to scan the screen. Scanning is an external operation, so the analogy process itself must be interrupted. Once again, the interruption is handled through sidegoaling. Operators within the analogy state signal the need to scan on the top state, and analogy is terminated. (The simplified representation in Figure 4 does not show this.) When the scan operation completes, the context still indicates a need for analogy, so analogy is proposed again. But now the information it needs is available, so scanning a second time is unnecessary.

5.3. BEHAVIOUR OF THE IDXL MODEL

To put these ideas together in a more comprehensible form, we now present a high-level step-by-step description of the model's behaviour on the Cricket Graph task. Initially the model has knowledge of Macintosh menu affordances, but for test purposes it does not begin with the knowledge that double-clicking an icon starts a program. Three runs of the model are described. (1) The model is given the task of starting Microsoft Word, which it learns by instruction. (2) It is then given the same task again and performs it by deliberate recall of the instructions. (3) Next it is given the task of starting Cricket Graph, which it performs by analogy to the Word task. Having started Cricket Graph, the model continues with its task and goes on to create a line graph.

To simplify this presentation, the model has been given the preferences of an aggressive explorer, so that it decides to take action after relatively little scanning. We will discuss in a later section how the model can produce more cautious behaviour.

5.3.1. Starting Microsoft Word

The task first given to the model is to start Microsoft Word. IDXL's default behaviour, with or without a task, is to visually scan its environment (here limited to the computer display screen). This is what it begins to do, selecting the scanning directions with experience-based preference knowledge. Almost immediately it sees the icon labelled "Cricket Graph". It attempts to comprehend this icon, but the comprehension operator's minimal effort returns only the fact that the icon matches nothing in the task description, so the scan continues. The next item identified is the icon labelled "Word". The model attempts to comprehend this, and the comprehend operator returns the fact that this label matches a key word in the task description.

Because this match is exact, the model describes to attempt further comprehension instead of scanning for other items. Its next attempt at comprehension suggests moving the mouse cursor to the icon. This is a safe, low-cost action, and the model's preference knowledge selects it over further scanning or comprehension. Now the cursor is pointed at the icon, but nothing has happened, so the model again enters the comprehension phase. Unfortunately, no more knowledge is forthcoming. In particular, the model tries to recall an affordance for the icon, and fails. It tries to generate potential answers to the question, "how do I start Word"? and this also fails. It then attempts to use analogy, but this too fails, because it has no prior example to draw on.

In the end, the model decides to ask for help, which it does by sidegoaling. The answer "double-click the word icon", is simulated. When the model perceives the answer, its top-state preference knowledge immediately decides that parsing this answer is the next best action, and that is done, forming a recognition chunk. Finally, with nothing else to do in the top state, the model drops back into comprehension. The comprehension operator can now use the newly parsed information, and it suggests double-clicking the icon as a safe physical action. Control again returns to the top state, the action is performed, and the Word program starts.

5.3.2. Restarting Microsoft Word

Now we restart the model and the simulation, but allow the model to retain the knowledge it has accumulated as chunks during the previous run. We give it the same task: Start Microsoft Word. The run proceeds much like before, although some of the comprehension work is performed by chunks formed during the first run. For example, the model immediately recognizes that the Cricket-Graph icon does not have a label match to the task. Still, it puts a quantum of comprehension effort into the item, and this time it concludes that there are no easily retrieved synonyms that would produce an indirect task-label match.

When the model sees the Word icon, the chunks formed in the previous run immediately identify the task-label match and suggest moving the mouse cursor to the icon, without further comprehension. But the model is still unable to recall the affordance for the icon. Once again, it generates a sequence of potential answers, and one of them is "Double-click the Word icon". The recognition chunk learned earlier fires, and the model recognizes that this is the action it was instructed to do. It learns a recall chunk, which will produce immediate recall the next time the task is presented. The information is also in working memory. The model again drops into the comprehend state, uses that information to suggest a physical action, and successfully starts the program.

5.3.3. Starting Cricket Graph

Next we give the model the task of starting Cricket Graph. The task has changed, so many of the chunks learned in the Word task no longer match. When the model first sees the Cricket Graph icon, it tries to comprehend it and recognizes a match. Further comprehension leads to the physical action of moving the mouse cursor to the icon. The model has learned no general procedure for starting programs, so it does not know that double-clicking the icon is the next thing to do. As in the Word example, when nothing is immediately recalled, the model tries generate-and-test of potential answers. Nothing is recognized, so now it tries analogy. This time analogy succeeds. The model recalls its experience with Word, and maps the information onto its current task. It learns a recognition chunk at this point, just as if it had been given instructions. And it starts Cricket Graph by double-clicking the icon.

5.3.4. Selecting Line from the Graph menu

The model's task description has two parts. It must start Cricket Graph, and it must create a line graph. In this run of the model, we have changed the task description slightly, so the second part is to create a "chart" graph. We have also given the model knowledge that "chart" and "line" are synonyms. This will allow us to demonstrate the model's behaviour in situations where the task and on-screen labels are not an exact match.

As soon as the Cricket Graph program begins, the model returns to its default scanning behaviour. Its background knowledge of Macintosh applications focuses its scan initially on the menu bar at the top of the screen. The first item perceived has the label "File", and a quick comprehension operation concludes that this is not a match to a key task word. This forms a chunk, and the model's preference knowledge leads its to continue its scan. It visually scans across the top menus, quickly noting that nothing matches, until it reaches the menu labelled "Graph". The first attempt at comprehension indicates that this is an exact match to a key word in the task, so the model drops again into the comprehend state, where it identifies a safe action related to the object: Move the mouse cursor onto it. Preference knowledge in the top state of the model prefers the action over further scanning or comprehension.

With the cursor on the item, the model decides to comprehend again. The result is another suggested action: press the mouse button. This is still relatively safe in the Mac interface, so the model's preference knowledge selects it. (A less exact match would have been temporarily abandoned at this point, while the model went on to scan other menu bar items.) The simulation now produces the pulldown menu, which causes a change in the model's visual field. Whenever such a sudden change occurs, the model has a high preference for shifting visual attention to the new item.

A scan operator performs this action, bringing the first item of the new menu into view. The comprehension operator is called on to consider the item, which has the label "Scatter". This is not an exact match to the task description, so scanning continues, this time moving through the items on the menu that has just dropped. (Again, experience-based preference knowledge helps guide the direction of scan.) The next item the model sees is labelled "Line". This is also an inexact match to the task as we have worded it, so the model passes over it after initial comprehension. The visual scan and comprehend sequence processes additional items in the menu, without finding a good match.

Now the scan preference knowledge causes the scan direction to reverse, so the model is passing over the same items again. Each time it sees an item for the second time, the comprehension operator is again selected, but this time the "no-match" result is immediately provided by the chunk formed during initial comprehension. Since the model's approach is to always do a small amount of comprehension work, it goes beyond the initial result and tries to identify synonyms that might link the task and label. For the Line label it identifies such a synonym: a synonym for "Line" (in the model's lexicon) is "Chart", and "Chart" is a word in the task.

If this were a direct match, the model would immediately try to identify a physical action that the item affords, and take that action. However, for a synonym the model is more cautious. It does not take action yet, but scans the other pulldown items again—one of them might also be a synonym. On returning to the Line item, it comprehends it again and decides to take the physical action of moving the mouse cursor onto it.

Having moved the mouse cursor, the model makes one final visual scan of the other items, rejecting each using the chunks previously formed, and then uses comprehension to suggest that releasing on the Line menu might advance its task goal. But the model recognizes that releasing on a menu item is a potentially expensive operation, so it enters the comprehension state yet again, in an attempt to predict or at least justify the action's result. The model notes that two task-goal words have been matched by labels on the current path: "Graph" and "Chart = Line". The model's background knowledge confirms that this indicates a good chance of success, so it performs the action. The simulation responds by presenting a simplified dialogue box, which the model deals with by using the same scan and comprehend process.

6. Evaluating IDXL

The approach of our research has been to identify behavioural patterns in the empirical protocols and produce a model that helps us to understand the mechanisms underlying those patterns. We have not made an effort to precisely match the model's behaviour to any one protocol. For the level at which the model operates, this would require far more data than we have available, both in terms of shifts of visual attention and the background knowledge of each user. However, the overall character of the model's behaviour is similar in important respects to the empirical data summarized in this paper. Additionally, the model produces many of the important aspects of behaviour reported for other models of display-based learning.

TABLE 1
Empirical data compared to behaviour of the IDXL model

Empirical data	IDXL model	Comments
Follows labels	Follows labels	
Exact label matches are acted on fastest	Exact label matches are acted on fastest	
Synonyms take longer	Synonyms take longer	
Inferences take the most time	*Current model does not handle labels that require inference*	Like analogy, inference would be a more costly (slower) incremental comprehension step
Meaningless labels require instruction	*Meaningless labels are not recognized by current model*	The model does take instruction, but not for meaningless labels
Poor labels and large number of objects interact to produce longer search times	Poor labels and large number of objects interact to produce longer search times	
Menu and dialogue box items easier to identify than direct-manipulation	*No difference*	
Repeated scans with iteratively deepening attention for poor labels	Repeated scans with incremental comprehension for poor labels	Incremental comprehension performs more time-consuming operations on later scans
Menu bar items are tried more quickly than pull-down menu items	Menu bar items are selected as soon as a label match is identified, pulldown items only after further comprehension	
Initial scanning is predominantly guided by menu layout (left-to-right, top-to-bottom, etc.)	Preferences cause scanning to be guided by menu layout	
Obviously incorrect labels are avoided	Visual scans cover all labels, but mouse is moved only to potential candidates	
Individuals vary in exploratory aggressiveness	Model's aggressiveness can be adjusted easily by changing preference knowledge	

6.1. EMPIRICAL DATA AND ALTERNATIVE VERSIONS OF THE MODEL

Table 1 summarizes important similarities and differences between the IDXL model and the empirical data described earlier in this paper. Comparing the model to the data analysed by Franzke (1994, 1995), we find that the high-level description of the behaviour in both cases reflects a label-following strategy. For both the model and the experimental subjects, labels that are perfect matches to a task word are selected most quickly. Synonyms require slightly longer in the empirical data, and the model requires another cycle of comprehension, not attempted until a second scan, to identify synonyms. In the empirical data, "inferences" require still longer. The model has no explicit comprehension operation called inferencing, but the more sophisticated cognitive processes it does use, such as analogy and thinking ahead, take more time to complete and are performed only after successive scans and simpler strategies fail. Finally, the empirical data showed that subjects are typically unable to discover items with labels having no clear connection to the task; they must receive instruction at this point. Similarly, the model includes no exhaustive search process, but it is prepared to ask for help when on-line exploration becomes too costly.

Franzke's analysis also showed that the number of objects on the screen interacts with label-following behaviour, especially in the case of poor labels. The IDXL model shows exactly this behaviour. Poor label matches cause the model to scan the display repeatedly before incremental comprehension identifies an appropriate control, and scan times are increased by the number of objects that must be considered. The third critical factor identified by Franzke's analysis was the type of affordance: Menu and dialogue box controls were easier to locate than direct-manipulation items. The model does not address this effect.

The analysis by Rieman (1994) restates many of these points at a lower level of detail. The key new finding in that analysis was iteratively deepening attention, a form of behaviour which the model exhibits. Items with labels that exactly match key words in the model's simplified goal representation are focused on immediately, while less perfect matches will attract repeated scans with increasingly time-consuming comprehension. Additionally, the empirical data showed that subjects will readily explore the system by pulling down menus from the menu bar, but are reluctant to take the additional step of releasing on items in those menus. The model shows the same balance. Pressing the mouse button to pull down menus from the top menu bar is done after only a few cycles of comprehension, while releasing on items within the pulldown menus requires more extended consideration, including thinking ahead to attempt to predict the effect of the action.

The model as described includes preference knowledge that represents the background experience of a single hypothetical individual. However, subjects with different backgrounds show considerable variety in their exploratory strategies. Similarly, small changes in the model's preferences can cause it to scan in different patterns, or to scan further before it decides to focus on an item, or to be more or less cautious about different physical actions. As an example, the version of the model we describe above pulls down the Graph menu as soon as its scan identifies the label, matching behaviour that was observed in only a few of the empirical protocols. By changing one item in the preference knowledge, we can produce a more cautious model, so that the entire top menu bar is scanned and Graph is considered more than once before any action is taken. Both versions can subsequently perform the task with no repeated scans.

6.2. EARLIER MODELS AND BEHAVIOUR NOT MODELLED

The IDXL model draws heavily on prior work. Like Ayn (Howes, 1994) and the ACT-R model described in Rieman (1994), it is committed to an approach in which large scale patterns of behaviour emerge out of local decisions. Like the ACT-R model, its fundamental behaviour is to scan the interface until it identifies a label that overlaps a word in its task. Like the Ayn model, it gradually accumulates display-based knowledge that leads to faster performance. It adds to these mechanisms the consideration of costs and benefits involved in menu selection, suggested by the model of Rehder *et al.* (1995). The model integrates these menu search processes with the instructional approach described in the TAL work (Howes & Young, in press) and the analogy model from Rieman *et al.* (1994).

6.2.1. Behaviour not modelled

The current version of the model is less complete in several areas than the work that preceded it. Although IDXL's scanning behaviour will eventually lead it to success in a simple menu-based situation, and it learns many recognition chunks that will speed later performance, it cannot use that information to avoid actions leading to failure or to prefer those that lead to success. The model needs additional strategies that will enable it to associate the outcomes of exploration with menu labels. These could be simple learning strategies that associate a label with "leads-to-failure", similar to the task-control knowledge learned by Ayn. It would, however, be more in keeping with IDXL's overall approach to associate system-specific semantic information with each label, such as a series of chunks of the form "label-Font-leads-to-Times" and "label-Font-leads-to-Helvetica". These would be retrieved and used by comprehension processes when the label was encountered on subsequent scans.

Also, the model performs only a small part of the Cricket Graph task. Modelling the interaction with a realistic simulation of the line-graph dialogue box requires further attention to the details of monitoring task completion and shifting attention from one part of a task description to another (Rieman, 1994). Similarly, the extended editing protocols in which the default line graph is modified require operators that produce a kind of means–ends behaviour (Newell & Simon, 1972), by comparing the default graph to the sample, identifying differences, and scanning for opportunities to reduce those differences.

In general, these operations can be conceptualized within the framework of the current model. Scanning would remain the default behaviour, and other cognitive activities would take place within the comprehension state, or as sidegoals from that state. However, many of the issues addressed by earlier work, such as the working memory constraints that help to shape Ayn's behaviour, would need to be reconsidered in this framework. Our intent here is not to claim that the additional modelling would be a trivial task, but to note that many forms of behaviour can fall within the scope of the basic approach of scanning and comprehending.

6.2.2. The CI model and errors

A significantly different approach to modelling the Cricket Graph task has been taken by Kitajima and Polson (1992, 1995), working in Kintsch's Construction–Integration framework (Kintsch, 1988). Unlike the IDXL system, the CI model does not define a single-item buffer for visual scanning. Instead, it initially considers all objects in the interface, then focuses on a subset that seem related to its task goals. That subset may be distributed physically across the display. The model provides a principled explanation of why even expert users make errors in display-based tasks. The model's most common form of error occurs when it is considering acting on a screen object, but fails to retrieve knowledge concerning the conditions associated with the correct action. At this point it will either select an incorrect action whose conditions are satisfied (i.e. double-clicking instead of releasing the mouse button), or choose to act on some other object (Kitajima & Polson, 1995: p. 90).

The IDXL label matching and synonym retrieval processes are simplifications of the CI work, and IDXL can also make errors if it decides to act before fully comprehending an item. For example, a slight shift in preference knowledge would cause IDXL to select the Line item the first or second time it saw it, even though its task was to create a "chart". The effect could be that Line was selected before the scan had a chance to reach an item labelled "Chart", if such an item existed. This kind of error reflects a novice user's insufficient caution in an unknown environment, while the CI errors reflect an expert's failure to use existing knowledge. Nevertheless, both forms of error arise when the respective models implicitly decide that the potential benefit of immediate action is preferable to the cost of further search—internal search of memory in the CI model, and a combination of internal and external search for IDXL.

7. Conclusions

At this point it will be useful to step back from the detailed description of the IDXL model. We summarize its structure and behaviour, discuss its implications for HCI, and compare it to related work in cognitive science.

7.1. SUMMARY OF THE IDXL MODEL

The IDXL model's default action, even without a defined task, is to scan its environment. When a new item is identified in the visual scan, the model attempts to "comprehend" it. The alternation between external scanning and internal comprehension allows the model to perform a dual-space exploration that gradually shifts its focus to a potentially productive route through an interface.

Comprehension is a general purpose operator, but its overall effect is to identify scanned objects and associated actions that might advance the model's task goals. A more complete version of the model would also acquire new goals this way. Comprehension is performed incrementally, and each pass can add knowledge to both working memory (e.g. the fact that the label being attended to matches a task word) and long-term memory (chunks). Many of the additions to long-term memory are small, situation-specific associations, which will support recognition in similar situations but not recall.

Extended comprehension strategies such as analogy are implemented as sidegoals, allowing the model to maintain relatively shallow goal stacks. This approach reflects the model's general commitment to interruptible processing.

Although the model's basic behaviour can be described simply as scanning and comprehending, the implementation and preferences associated with those operations require a great deal of background knowledge. The knowledge contained in the model we have developed is specific to the Macintosh interface. It includes preferences for directions to scan, preferences for deciding between scanning and comprehension, and specific strategies and detailed facts used in comprehension.

The current model leaves several areas undefined, including learning to avoid menus containing only useless items, sophisticated monitoring of task progress, and setting and terminating goals. Our underlying theoretical approach suggests that these would be implemented as comprehension operations, which, in turn, are called in response to perceptual data gathered through the basic scanning behaviour of the model.

7.2. IMPLICATIONS FOR HCI

For the purposes of interface design, the model and the empirical data reviewed earlier combine to emphasize the factors that will shape users' exploratory behaviour. The task description will strongly limit the set of interface objects that are considered for action. And no exploratory action will be attempted until the user has balanced its predicted "safety" against the quality of its label. The safest actions are those whose effect is predictably minimal or trivial to undo, such as dropping a pulldown menu. The user may perform such actions as soon as an approximate label match is identified. "Unsafe" actions, on the other hand, have unpredictable and potentially permanent effects. Examples include releasing on a pulldown menu item, or clicking "OK" in a dialogue box. A user will be reluctant to take these actions unless the label is very good or background knowledge clearly justifies the item's relationship to the task.

Of course, choosing "good" labels is not a trivial design exercise. Furnas, Landauer, Gomez and Dumais (1987) have shown that different users will use different names for any given feature. Those results describe recall knowledge, but similar individual differences may apply to label recognition, especially where the user chooses to avoid a label in the mistaken belief that it identifies an unsafe and totally irrelevant action. If this is the case, then some users of any system may occasionally be blocked from attempting actions essential to their task, especially at the level of selections from the pulldown menus. Application designers need to be aware of points where blockage may occur, and provide some kind of assistance to overcome these problems. Traditional ways to provide this assistance are in the form of on-line help or printed manuals, or alternative routes such as toolbars.

An additional approach suggested by the current research would be to supply additional information that could shift the balance at the point where benefit-to-cost comparisons are most difficult. The model and data show that users of WIMP interfaces like the Macintosh will quickly scan the menu bar and pulldown menus, but will be slow to select items from the pulldown menus. Pointing the mouse at those items, however, is a safe behaviour that will occur. On later passes, the comprehension phase will be longer, and the mouse cursor may pause for some time over an item of interest. An application could recognize this pause and temporarily change the system state to what it would be if the user selected the item under the cursor—typically, by displaying a dialogue box. This change might be visibly marked in some way, perhaps by appearing in grayed-out form, and it could be immediately undoable simply by moving the mouse off the item. This approach would safely provide the additional information the user needs. (A related approach is used in some recent commercial products, which display a help message for cryptic toolbar icons if the user pauses the mouse cursor over them.)

A larger issue that the model and data both demonstrate is that some actions, no matter how obvious to the designer, will simply not be discovered by some users. In this light, manuals, on-line help, and other support facilities take on a new meaning: they allow exploration beyond the invisible bounds of "safe" trial and error.

7.3. THE IDXL APPROACH AND COGNITIVE SCIENCE

Although the IDXL model and the empirical data that inform it are closely tied to the Macintosh environment, the model describes a form of behaviour that might be found in many situations where highly trained task-specific skills are lacking. Examples are easy to find: locating the light switch in a rented car; assembling do-it-yourself furniture; deciding whether to sit or wait to be seated in a new restaurant; opening an unfamiliar sealed food container. In each of these situations, an individual applies a combination of background knowledge, perceptual cues, and exploratory actions. And those actions, although many will lead to unpredictable results, must be considered in terms of their potential costs and benefits, where benefits include not only performance but also information gathering. (Kirsh & Maglio, 1994, raise closely related issues.)

With this interactive, unplanned, weakly goal-driven behaviour, IDXL is one of several recent information-processing systems that respond implicitly or explicitly to the criticisms raised by the situated-cognition approach (Suchman, 1987; Vera & Simon, 1993; and associated discussion in Greeno, 1993). Suchman's terminology is especially appropriate for describing certain features of IDXL. The model uses processes such as analogy and thinking ahead as "resources" to guide its actions. The model's approach of scanning until interrupted by comprehension, which can itself be interrupted, has strong similarities to the layered approach described by Brooks (1991), which is also designed to deal with dynamic environments. In both systems, unplanned but task-oriented behaviour emerges from interactions between the environment and the agent. The IDXL modelling effort affirms that such "situated action" is often appropriate.

However, the modelling work also emphasizes the need for an understanding of the information-processing aspects of behaviour. As we have emphasized, exploratory learning is a phenomenon that can be productively described in terms of two spaces, an external space of interactions and an internal space of knowledge, strategies, and task goals. A common problem-solving approach, in the Macintosh environment and many others, will be to search incrementally deeper into each of these spaces until the task is accomplished or the driving goals change. This is behaviour that cannot be understood without considering the dynamic nature of both the external and the internal worlds.

Clayton Lewis, Peter Polson, Marita Franzke, and Steve Payne contributed to many of the discussions leading to the work described here. We thank Frank Ritter and an anonymous referee for valuable comments on the draft manuscript. Mike Hucka's support for the SDE system greatly aided the modelling. This work was funded by the UK Joint Councils Initiative in Cognitive Science and Human–Computer Interaction. The empirical research was supported by grant number IRI-9116640 from the National Science Foundation.

References

Anderson, J. R. (1990). *The Adaptive Character of Thought.* Hillsdale, NJ: Lawrence Erlbaum Associates.

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

Brooks, R. A. (1991). Intelligence without representation. *Artificial Intelligence,* **47,** 139–159.

Carroll, J. M. (1982). The adventure of getting to know a computer. *IEEE Computer,* **15,** 49–58.

Caroll, J. M. (1990). *The Nurnberg Funnel.* Cambridge, MA: MIT Press.

Carroll, J. M. & Mazur, S. A. (1986). Lisa Learning. *IEEE Computer,* **19,** 35–49.

Carroll, J. M., Mack, R. L., Lewis, C. H., Grischkowsky, N. L. & Robertson, S. P. (1985). Exploring a word processor. *Human–Computer Interaction,* **1,** 283–307.

Charney, D., Reder, L. & Kusbit, G. (1990). Goal setting and procedure selection in acquiring computer skills: a comparison of tutorials, problem solving, and learner exploration. *Cognition and Instruction,* **7,** 323–342.

Engelbeck, G. E. (1986). *Exceptions to generalizations: implications for formal models of human–computer interaction.* Masters Thesis, Department of Psychology, University of Colorado, Boulder.

Franzke, M. (1994). *Exploration, acquisition, and retention of skill with display-based systems.* Ph.D. Thesis, Department of Psychology, University of Colorado, Boulder.

Franzke, M. (1995). Turning research into practice: characteristics of display-based interaction. *Proceedings of the Conference on Human Factors in Computing Systems,* pp. 421–428. New York, NY: Association for Computing Machinery.

Franzke, M. & Rieman, J. (1993). Natural training wheels: learning and transfer between two versions of a computer application. In T. Grechenig & M. Tscheligi, Eds. *Lecture Notes in Computer Science, Human Computer Action: Proceedings of the Vienna Conference on HCI,* pp. 317–328. Berlin: Springer-Verlag.

Furnas, G. W., Landauer, T. K., Gomez, L. M. & Dumais, S. T. (1987). The vocabulary problem in human-system communication. *Communications of the ACM,* **30,** 964–971.

Greeno, J. G. (Ed.) Special issue: situated action. *Cognitive Science,* **17,** 1–133.

Holland, J., Holyoak, K., Nisbett, R. & Thagard, P. (1986). *Induction: Processes of Inference, Learning, and Discovery.* Cambridge, MA: MIT Press.

Howes, A. (1994). A model of the acquisition of menu knowledge by exploration. *Proceedings of CHI'94 Conference on Human Factors in Computer Systems,* pp. 445–451. New York, NY: Association for Computing Machinery.

Howes, A. & Young, R. M. (in press). Learning consistent, interactive, and meaningful task-action mappings: a computational model. *Cognitive Science.*

Huffman, S. (1993). Learning from instruction: a knowledge-level capability within a unified theory of cognition. *Proceedings of the 15th Meeting of the Cognitive Science Society.* Boulder, CO, June 1993.

Huffman, S. (1994). *Instructable autonomous agents.* Ph.D. Thesis, Department of Computer Science and Engineering, University of Michigan, MI, U.S.A.

Kintsch, W. (1988). The role of knowledge in discourse comprehension: a construction–integration model. *Psychological Review,* **95,** 163–182.

Kirsh, D. & Maglio, P. (1994). On distinguishing epistemic from pragmatic action. *Cognitive Science,* **18,** 513–549.

Kitajima, M. & Polson, P. G. (1992). A computational model of skilled use of a graphical user interface. *Proceedings of CHI'92 Conference on Human Factors in Computing Systems,* pp. 241–249. New York, NY: ACM.

Kitajima, M. & Polson, P. G. (1995). A comprehension-based model of correct performance and errors in skilled, display-based human computer interaction. *International Journal of Human–Computer Studies* **43,** 65–99.

Klahr, D. & Dunbar, K. (1988). Dual space search during scientific reasoning. *Cognitive Science,* **12,** 1–48.

Korf, R. E. (1985). Depth-first iterative deepening: an optimal admissible tree search. *Artificial Intelligence,* **27,** 97–109.

Korf, R. E. (1988). Search: a survey of recent results. In H. E. Shrobe, and the Association for Artificial Intelligence, Eds. *Exploring Artificial Intelligence.* pp. 197–237. San Mateo, CA: Morgan Kaufmann.

Laird, J., Newell, A. & Rosenbloom, P. (1987). SOAR: an architecture for general intelligence. *Artificial Intelligence,* **33,** 1–64.

Laird, J. E. & Rosenbloom, P. S. (in press). The evolution of the Soar cognitive architecture. In T. Mitchell, Ed. *Mind Matters.* Hillsdale, NJ: Lawrence Erlbaum Associates.

Lewis, C. (1988). Why and how to learn why: analysis-based generalizations of procedures. *Cognitive Science,* **12,** 211–256.

Lewis, C. H., Polson, P. G., Wharton, C. & Rieman, J. (1990). Testing a walkthrough methodology for theory-based design of walk-up-and-use interfaces. *Proceedings of CHI'90 Conference on Human Factors in Computer Systems.* pp. 235–241. New York, NY: Association for Computing Machinery.

Malone, T. W. (1982). Heuristics for designing enjoyable user interfaces: lessons from computer games. *Proceedings of the Conference on Human Factors in Computing Systems,* pp. 63–68. New York, NY: Association for Computing Machinery.

Mannes, S. M. & Kintsch, W. (1991). Routine computing tasks: planning as understanding. *Cognitive Science,* **15,** 305–342.

May, J., Scott, S. & Barnard, P. (1995). *Structuring Displays: A Psychological Guide.* Geneva: Eurographics Tutorial Notes Series, EACC.

Newell, A. (1990). *Unified Theories of Cognition. The William James Lectures.* Cambridge, MA: Harvard University Press.

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

Payne, S. J., Squibb, H. R. & Howes, A. (1990). The nature of device models: the yoked state space hypothesis and some experiments with text editors. *Human–Computer Interaction,* **5,** 415–444.

Polson, P. G. & Lewis, C. H. (1990). Theory-based design for easily learned interfaces. *Human–Computer Interaction,* **6,** 19–220.

Polson, P. G., Lewis, C., Rieman, J. & Wharton, C. (1992). Cognitive walkthroughs: a method for theory-based evaluation of user interfaces. *International Journal of Man–Machine Studies,* **36,** 741–773.

Rehder, B., Lewis, C., Terwilliger, B., Polson, P. & Rieman, J. (1995). A model of optimal exploration and decision making in novel interfaces. *Conference Companion to CHI'95 Conference on Human Factors in Computer Systems,* pp. 230–231. New York, NY: Association for Computing Machinery, pp. 230–231.

Rieman, J. (1993). The diary study: a workplace-oriented tool to guide laboratory studies. *Proceedings of InterCHI'93 Conference on Human Factors in Computer Systems.* pp. 321–326. New York, NY: Association for Computing Machinery.

Rieman, J. (1994). *Learning strategies and exploratory behavior of interactive computer users.* Ph.D. Thesis, Technical Report CU-CS-723-94, Department of Computer Science, University of Colorado, CO, U.S.A.

Rieman, J. (in press). A field study of exploratory learning strategies. *ACM Transactions on Computer–Human Interaction.*

Rieman, J., Lewis, C., Young, R. M. & Polson, P. G. (1994). "Why Is a Raven Like a Writing Desk?" Lessons in Interface Consistency and Analogical Reasoning from Two Cognitive Architectures. *Proceedings of CHI'94 Conference on Human Factors in Computer Systems.* pp. 438–444. New York, NY: Association for Computing Machinery. pp. 438–444.

Rieman, J., Davies, S., Hair, D. C., Esemplare, M., Polson, P. G. & Lewis, C. (1991). An automated walkthrough. *Proceedings of CHI'91 Conference on Human Factors in Computer Systems.* pp. 427–428. New York, NY: Association for Computing Machinery.

Rosenbloom, P. S., Laird, J. E. & Newell, A. (1991). A preliminary analysis of the SOAR architecture as a basis for general intelligence. *Artificial Intelligence,* **47,** 289–326.

SHNEIDERMAN, B. (1983). Direct manipulation: a step beyond programming languages. *IEEE Computer,* **16,** 57–69.

SHRAGER, J. (1985). Instructionless learning: discovery of the mental model of a complex device. Ph.D. Thesis, Department of Psychology, Carnegie Mellon University, Pittsburgh, PA, U.S.A.

SHRAGER, J. & KLAHR, D. (1986). Instructionless learning about a complex device: the paradigm and observations. *International Journal of Man–Machine Studies,* **25,** 153–189.

SIMON, H. A. (1973). The structure of ill-structured problems. *Artificial Intelligence,* **4,** 181–201.

SUCHMAN, L. A. (1987). *Plans and Situated Actions.* Cambridge: Cambridge University Press.

VERA, A. H., LEWIS, R. L. & LERCH, F. J. (1993). Situated decision-making and recognition-based learning: applying symbolic theories to interactive tasks. *Proceedings of the 15th Annual Conference of the Cognitive Science Society,* Boulder, CO, June 1993.

VERA, A. H. & SIMON, H. A. Situated action: a symbolic interpretation. *Cognitive Science,* **17,** 7–48.