HUMAN-COMPUTER INTERACTION, 1987-1988, Volume 3, pp 223-274
Copyright © 1987-1988, Lawrence Erlbaum Associates, Inc.

A Keystroke Analysis of Learning and Transfer in Text Editing

Mark K. Singley and **John R. Anderson**
Carnegie-Mellon University

ABSTRACT

Two experiments studied the acquisition and transfer of text-editing skill. The first experiment, originally reported in Singley and Anderson (1985) but reanalyzed in greater detail here, found nearly total transfer between two similar line editors and partial transfer from the line editors to a screen editor. Analyses of the keystroke data revealed that the majority of the improvement during both learning and transfer was concentrated in the planning components of the skill. The second experiment found little evidence for negative transfer between a pair of screen editors designed for maximal interference using a classic interference paradigm. The few instances of negative transfer observed were better characterized as the positive transfer of nonoptimal methods rather than instances of true procedural interference. These results support an identical elements model of transfer based on a production system representation of cognitive skill. The relative magnitudes of transfer observed were consistent with detailed measures of production system overlap. In addition, localized transfer sites were hypothesized and identified through a series of microanalyses. Finally, specific transfer predictions based on the differential practice of general and specific components were tested and confirmed.

Authors' present addresses Mark K Singley and John R. Anderson, Department of Psychology, Carnegie-Mellon University, Pittsburgh, PA 15213.

CONTENTS

1. INTRODUCTION
2. EXPERIMENT 1: IDENTICAL ELEMENTS IN TEXT EDITING
    2.1. Method
    2.2. Results
        Summary of Macroanalysis
        Microanalysis of Learning and Transfer
        First Learning Analysis: Relative Contributions of Planning Time and Execution Time
        Second Learning Analysis: Division Into LL/MT
        Third Learning Analysis: Division Into Methods
        First Transfer Analysis: LL/MT
        Second Transfer Analysis: MT Methods
    2.3. Summary of Experiment 1
3. EXPERIMENT 2: IDENTICAL ELEMENTS AND NEGATIVE TRANSFER
    3.1. Method
    3.2. Results
        Macroanalysis of Transfer
        Microanalysis of Transfer
        First Transfer Analysis: LL/MT
        Second Transfer Analysis: Division Into Methods
        Third Transfer Analysis: MT Methods
    3.3. Summary of Experiment 2
4. SIMULATION MODELS AND QUANTITATIVE PREDICTIONS
5. GENERAL DISCUSSION
APPENDIX: KEYSTROKE PARSING PROCEDURE

1. INTRODUCTION

The aim of our research is to take a detailed look at transfer among relatively complex cognitive skills. In particular, we have studied how people transfer among various text editors. Issues of transfer have fallen in and out of the main focus of psychology several times in the past. Perhaps the first psychologist to study transfer was Thorndike at the turn of the century (Thorndike & Woodworth, 1901). Thorndike took issue with the prevailing opinion concerning education during his time, namely, the "Doctrine of Formal Discipline" (Angell, 1908). This doctrine claimed that studying otherwordly subjects such as Latin and geometry were beneficial because they served to discipline the mind. "Formal Discipline" viewed the mind as a collection of general faculties, such as observation, attention, discrimination, and reasoning, which were exercised in much the same way as a set of muscles. The content of the exercise made little difference; most important

was the level of exertion (hence the fondness for Latin and geometry). Transfer in such a view is broad and takes place at a general level, sometimes spanning domains that share no content. For example, training in chess should transfer to computer programming because both skills involve the use of the general reasoning faculty. This view has taken on a variety of forms over the years and has been advanced recently by those who champion LOGO programming as a powerful new vehicle for teaching general problem solving (Linn & Fisher, 1983; Papert, 1980).

Thorndike performed a series of experiments which showed that transfer was much narrower in scope than was predicted by the "Doctrine of Formal Discipline." In one experiment, no correlation was found between memory for words and memory for numbers. In another, accuracy in spelling was not correlated with accuracy in arithmetic (Orata, 1928). To explain these and other similar results, Thorndike proposed the Theory of Identical Elements, which stated that training in one kind of activity would transfer to another only if the activities shared common elements. However, Thorndike's theory was not explicit about the respresentation of cognitive skills, so it was quite difficult to identify just what these elements were. In fact, there has been considerable debate concerning the exact nature of these elements, including the meaning of the troublesome term *identical* (Ellis, 1965; Orata, 1928). Just how identical did the elements have to be? Are subjects able to generalize across any dimensions? For example, could someone pound nails with a red hammer after having learned with a green? Thorndike attempted rather unsuccessfully to answer these questions, but quite simply, did not have the formal and theoretical tools to develop his ideas properly. However, it is generally accepted (Bower & Hilgard, 1981; Orata, 1928) that Thorndike's elements were quite specific and tied to a superficial analysis of the skill, which was consistent with his behaviorist outlook. Thus, Thorndike and the proponents of the "Doctrine of Formal Discipline" sat at opposite endpoints of the transfer-specificity continuum.

As just mentioned, one of Thorndike's problems was the lack of an adequate theory of knowledge representation in which he could define identical elements. Recently, however, his theory has been recast in terms of various modern formalisms and applied with reasonable success by some researchers. One factor that distinguishes these modern efforts from Thorndike's is that the elements are not necessarily tied to superficial behaviors but rather can be defined in purely cognitive terms. This allows for the sharing of abstract mental objects such as goal structures and plans among different procedures. This means that current analyses of transfer sometimes predict greater levels of transfer than what might be predicted given superficial analyses of similarity.

In one such attempt to revive the Theory of Identical Elements, Moran (1983) developed a task analysis technique called *External Task, Internal Task*

(ETIT) that conceptualizes the use of a device such as a text editor as a mapping from the user's goals and intentions (the external task) into the operating principles or semantics of the device (the internal task). One can use this technique to predict transfer among devices by merely gauging the differences between the mappings for different devices. One performs set computations on the two sets of mapping rules, and the degree of overlap predicts the degree of transfer. As Moran pointed out, this work was at an early stage. However, ETIT has already been used to account for errors made by novices learning a screen editor (Douglas & Moran, 1983) and the fact that novices can perform certain tasks they have not been taught (Douglas, 1983).

More recently, Polson and Kieras (1985) used a similar kind of analysis to predict the amount of transfer between different text-editing procedures. They identified production rules as the elements of skill and consequently used production set overlap as a predictor of transfer. They found that the amount of time it takes to learn a new procedure is primarily a function of the number of new production rules in that procedure. By varying the serial position of a procedure within a training set of similar procedures, dramatic differences in learning times were predicted and observed. In another study, Kieras and Bovair (1985) found similar results with subjects learning to use a simple control panel. Once again, the total time to learn a new procedure was predicted quite well by the number of new productions.

A potential problem with any identical elements account of transfer is its treatment of negative transfer. At first blush, one might expect that the worst possible transfer situation in such models is zero transfer, which occurs when there is no overlap between elements. However, if one adopts condition-action rules as the elements of cognitive skill, it is possible to account for negative transfer as the firing of rules whose conditions match in the transfer task but whose actions are inappropriate or nonoptimal. Of course, such a phenomenon complicates identical elements models, because it means there is not necessarily a straightforward mapping between the proportion of common elements and the overall level of performance. In one of the experiments reported here, we explore the phenomenon of negative transfer in text editing and discuss possible implications for identical elements models.

We chose text editing as our domain for a variety of reasons, but perhaps most important was the existence of a well-specified theory of expert performance. Card, Moran, and Newell (1983) put forth a series of information-processing models at various levels of detail that account for an impressive percentage of error-free, expert behavior. These models are useful in that they serve as a well-defined endpoint for skill acquisition in text editing.

The goal of our research is to use these models to extend the identical elements theory of transfer. The models supply us with a representation of text-editing skill that can be used to identify the elements common to different editors. More specifically, the models provide us with the abstract goal

structures that underlie and organize the production of keystrokes in the editors. Our predictions of transfer between the editors hinge on the fact that, even though the specific commands used to accomplish edits in the various systems may be different, the underlying goal structures are largely the same and provide the basis for positive transfer.

Like Polson and Kieras (1985), a major assumption in our work is that single production rules are the units of cognitive skill—the elements that Thorndike was searching for. What we are proposing is essentially a modern version of the Theory of Identical Elements based on productions rather than stimulus-response bonds. Unlike Thorndike's superficial elements, productions are versatile and powerful computational entities. It is well known that production systems have the computational power of Turing machines. Productions are abstract and can be used to represent many different problem-solving methods at various levels of generality. Productions are often used to represent cognitive processes that have no direct impact on the external world yet, nonetheless, play an important role in skilled behavior like planning and problem decomposition (i.e., subgoaling). Therefore, not only external but also internal "actions" are considered in calculations of transfer. In our models, all the components of skill (e.g., goal structures, methods, operators, decision rules, and sequences of user actions) are represented as productions.

Aside from our emphasis on task analysis and production system simulation, our work differs from previous work on transfer as follows:

Multiple Trials. Many studies have probably underestimated transfer because of a failure to insure that something was learned in the first place (Hayes & Simon, 1977; Smith, 1986). It is not surprising that transfer is minimal when subjects have only a single trial in both the base and transfer tasks. To better understand both learning and transfer, we trace our subjects' performance over multiple trials. We feel this methodology yields more sensitive measures of transfer than single trials in much the same way as time to relearn is a more sensitive measure of retention than recall or recognition.

Fine-Grained Analyses. In any account of learning and transfer, the level of analysis is critical. Central to our effort is the goal of determining the loci of transfer at the finest grain of analysis possible. Most desirable would be a grain size that allowed for the separation and independent measurement of all learning and transfer components. This would allow the most rigorous test of any common elements theory of transfer. However, given imperfect theories of skill representation and imperfect behavioral measures, this is presently and perhaps forever impossible. Unable to determine a priori the proper grain of analysis, we have conducted a series of analyses at several different grain sizes. We hope to understand learning and transfer by successive approximation, using the more aggregate measures to guide our analysis at more detailed levels.

2. EXPERIMENT 1: IDENTICAL ELEMENTS IN TEXT EDITING

The results of the first experiment were originally reported in Singley and Anderson (1985), but are analyzed in much greater detail here. The experiment involved teaching groups of computer-naive subjects one or two line editors and then a screen editor. Of particular interest was the following.

The Magnitude of Transfer Among Different Editors. Would transfer be positive, negative, or nonexistent? How would transfer among the line editors compare with transfer from the line editors to the screen editor? A production system analysis of the structural similarity of the editors suggested that, whereas the goal structures of the line editors were quite similar, the line editors and the screen editor shared few elements. Perhaps the magnitudes of transfer would reflect this difference.

Identification of Learning and Transfer Components. Could transfer effects be localized to particular subskills and pieces of knowledge? What would be the nature of these components?

2.1. Method

This section is a somewhat abbreviated version of that appearing in Singley and Anderson (1985), which should be consulted for additional details.

Subjects. Subjects were 24 women between the ages of 18 and 30 from a local secretarial school. None of the subjects had any computer experience, but all could type proficiently.

Materials. Subjects learned from a set of three commercially available text editors. Two of these editors, UNIX *ED* (Kernighan & Pike, 1984) and VMS *EDT* ("Introduction to the EDT editor," 1982), belong to the genre known as line editors, whereas the third editor, UNIX *EMACS* (Gosling, 1981), belongs to the genre known as screen editors.

Subjects were taught a minimum core set of commands for each editor. These commands were totally sufficient for the kinds of edits our subjects had to perform. In the line editors (*ED* and *EDT*), the core set included commands for: (a) printing, deleting, inserting, and replacing lines; and (b) substituting strings within lines (the substitution command also provided for string insertion and deletion). In the screen editor (*EMACS*), the core set included commands for: (a) moving the cursor forward, backward, up, and down; and (b) deleting characters, words, and strings. Of course, the character of the commands differed markedly between the line editors and the screen editor. The majority of *EMACS* commands pertained to moving the cursor and involved special terminal keys. In all editors, subjects were spared from learning the procedures for reading and writing files and were instead fed files automatically by experimental software. See Figure 1 for a listing of the core set of commands for each of the editors.

Subjects edited sections of a book on cognitive psychology that resided on a local computer system. The book was sectioned into 18-line files, and each file was randomly mutilated by a text-mutilation program. The program performed 6 of the 12 possible mutilations on each file. The 12 mutilations were defined by crossing the editing operations insert, delete, and replace by the data objects character, word, string, and line. It took two files to cover all 12 mutilations; the same mutilations occurred on every other screen. Each file constituted a single trial.

The subjects' task was to correct the errors introduced by the mutilation program. They worked from a marked-up copy of the files placed in a loose-leaf binder (see Figure 2 for a sample page). The binder sat on the table beside the terminal. Each page corresponded to a single file.

Design. The study used a 2×2 between-subjects design with two control groups. The first factor was number of line editors learned (one vs. two), and the second was initial line editor (*ED* vs. *EDT*). The two control groups learned no line editors; their first exposure to text editing was *EMACS*. One of the control groups spent the entire experiment editing with *EMACS*. (This was the control that would reveal whether transfer to *EMACS* from the line editors was positive or negative.) The other control group practiced typing at the terminal prior to editing with *EMACS*. This group typed for the amount of time the experimental groups spent learning the line editors. (This was the control that would reveal the perceptual–motor component of transfer.)

Procedure. The experiment consisted of 6 consecutive days of text editing. Each day consisted of a 3-hr session interrupted by two 10-min breaks after the first and second hours. Subjects were run by pairs in a quiet experimental room.

On the first day, all subjects (except those in the typing control condition) were given a brief introduction to the set of commands they would be using that day. This introduction consisted of a brief description of each command followed by a demonstration on the terminal. This introduction lasted approximately 30 min.

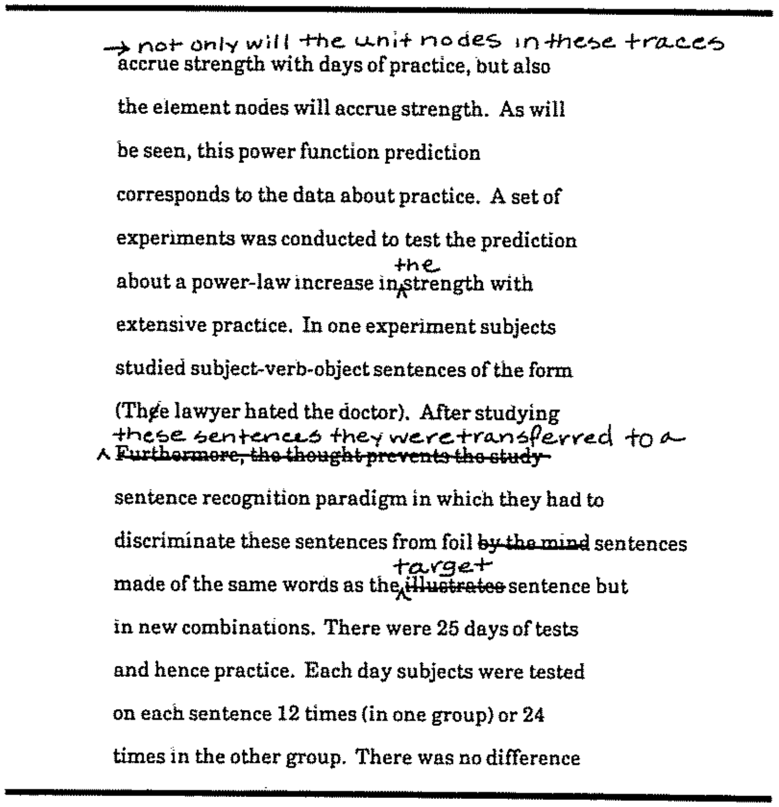
The subjects then began editing at the terminals. An experimenter was present in the room at all times to answer any questions or help with particularly difficult problems. Experimenters were told not to intervene unless a subject asked for help. A single tutor was designated for each editor, so that all subjects' experiences with a single editor would be similar. As experimenter was totally confounded with editor in the experiment, the results should not be regarded as a totally valid comparison of the editors.

The subjects spent the first 2 days practicing their first editor. On the third

Figure 1. **Command summary for the three editors. In the figure, ↑denotes a control character and] an escape character.**

Command Type	Editor	Command	Action
Locative	*ED*	1,$p	prints all lines of the file
		3p	prints the third line
		.p	prints the current line
		=	prints the line number of the current line
		RETURN	prints the line following the current line
	EDT	t whole	prints all lines of the file
		t 'dog'	prints the first line following the current line that contains 'dog'
		t – 'dog'	prints the first line before the current line that contains dog
		t	prints the current line
		DELETE	prints the line following the current line
	EMACS	↑f	moves cursor forward one character
		]f	moves cursor forward one word
		↑b	moves cursor backward one character
		]b	moves cursor backward one word
		↑a	moves cursor to beginning of line
		↑e	moves cursor to end of line
		↑p	moves cursor to previous line
		↑n	moves cursor to next line
Mutative	*ED*	.a	inserts lines after the current line (type '.' to exit the insert mode)
		.d	deletes the current line
		.c	replaces the current line (type '.' to exit the insert mode)
		s/a/b/p	substitutes the first occurrence of 'a' with 'b' on the current line
	EDT	i	inserts lines after the current line (type ↑z to exit the insert mode)
		d	deletes the current line
		r	replaces the current line (type ↑z to exit the insert mode)
		s/a/b	substitutes the first occurrence of 'a' with 'b' on the current line
	EMACS	↑d	deletes the character marked by the cursor
		]d	deletes the word marked by the cursor
		DELETE	deletes the character to the left of the cursor
		↑k	deletes from the current cursor position to the end of the line
		a	inserts the character 'a' at the current cursor position (EMACS is in insert mode by default)

Figure 2. **Sample page of corrections.**



→ not only will the unit nodes in these traces
accrue strength with days of practice, but also

the element nodes will accrue strength. As will

be seen, this power function prediction

corresponds to the data about practice. A set of

experiments was conducted to test the prediction

about a power-law increase in the strength with

extensive practice. In one experiment subjects

studied subject-verb-object sentences of the form

(The lawyer hated the doctor). After studying

these sentences they were transferred to a
~~Furthermore, the thought prevents the study~~

sentence recognition paradigm in which they had to

discriminate these sentences from foil ~~by the mind~~ sentences

made of the same words as the target ~~illustrates~~ sentence but

in new combinations. There were 25 days of tests

and hence practice. Each day subjects were tested

on each sentence 12 times (in one group) or 24

times in the other group. There was no difference

day, those subjects in the two-line editor conditions switched to their second editor (either *ED* or *EDT*), whereas the other subjects remained on their first. However, all subjects received a second introduction to the set of commands they would be using. (This constituted a review for the subjects who did not switch.) In this way, the amount of formal instruction received by subjects was constant across conditions.

On the fifth day of the experiment, all experimental subjects and the typing control group transferred to *EMACS*. After receiving formal instruction on the commands, these subjects spent the last 2 days practicing *EMACS*. (The *EMACS* control group spent all 6 days learning *EMACS*. They received formal instruction on the first, third, and fifth days.)

Those subjects in the typing control group spent the first 4 days typing the manuscript that the experimental groups were editing. In addition to incorporating all the corrections marked on the manuscript, subjects had to correct typing mistakes as they were made. This rule was enforced by a program that checked the stream of keystrokes against a target file and deactivated the keyboard once a difference was detected. Subjects could only reactivate the keyboard by pressing the delete key, which erased the mistake. This practice resulted in a level of frustration similar to that experienced by subjects in the experimental conditions. Thus, the typing control group had experience reading the manuscript, interpreting the edits, and interacting through the terminal.

Keystroke data accurate to within 1 sec were collected for all subjects. In addition, the edited versions of the mutilated files were saved to allow for error checking.

2.2. Results

In this section, we first quickly summarize subjects' performance in terms of gross measures, such as time per edit and keystrokes per trial. This macrolevel of analysis, which was the stopping point in Singley and Anderson (1985), gives us a general understanding of what is happening and encourages us to explore certain phenomena further. The bulk of this section is devoted to a detailed analysis of the keystroke data. In this microanalysis, we hope to localize the components of learning and transfer intimated at the coarser level.

Summary of Macroanalysis

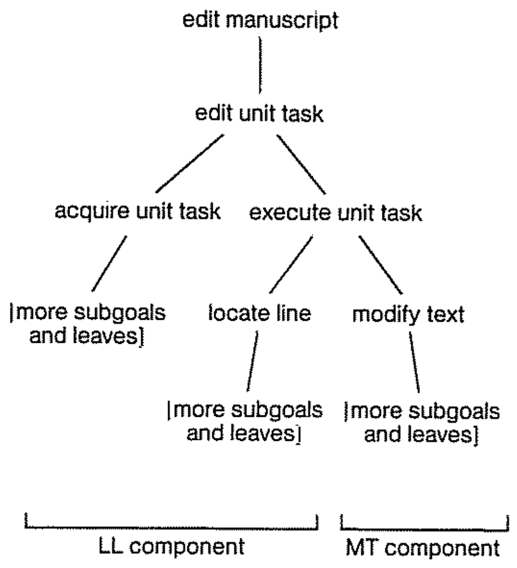
The macroanalysis (Singley & Anderson, 1985) presented results purely in terms of global measures. However, the basic outlines of the phenomena of learning and transfer emerged. In terms of both time per trial and keystrokes per trial, we observed the following:

1. A consistent rank ordering of the editors (*EMACS, ED, EDT*) in terms of learnability and ease of use, with *EMACS* being the easiest and *EDT* the hardest.
2. Near total transfer between the two line editors. For example, 2 days of practice on *ED* was nearly as good as 2 days of practice on *EDT* in terms of preparation for further editing with *EDT*.
3. Moderate amount of transfer from the line editors to *EMACS*.
4. Slight transfer from the typing control condition to *EMACS*.
5. No sign of negative transfer. All transfer at the macrolevel was overwhelmingly positive.

Microanalysis of Learning and Transfer

The goal of the microanalysis is to isolate and independently measure the acquisition and transfer of the various elements of text-editing skill. To do this, we need some theory of performance in text editing to identify the

Figure 3. **The top-level goal structure of text editing.**



theoretically significant components. As mentioned previously, an extensive task analysis of text editing already exists in the work of Card et al. (1983). According to their GOMS formulation, text editing consists of a series of largely independent unit tasks, each of which is accomplished through the satisfaction of three subgoals: encode the edit from the manuscript (acquire-unit-task), move to the line requiring modification, and modify the text (see Figure 3). These three subgoals are instances of goals, one of the four components of skill in the GOMS model. The other components are operators, methods, and selection rules.

As an initial goal of the microanalysis, we set out to trace the acquisition and transfer of the various unit tasks of text editing and their major subgoals: acquire-unit-task, locate-line, and modify-text. We do this by taking a detailed look at the stream of keystrokes executed by our subjects over the course of the experiment. The keystroke data can be viewed as a series of keystroke bursts separated by pauses. These bursts and pauses have psychological significance in that the pauses represent the mental preparation time for an operation and the bursts represent its execution. Presumably, as a person becomes more skillful at text editing, the frequency and duration of the bursts and pauses change, reflecting the acquisition and compilation of the various components of text-editing knowledge.

Keystroke Parsing. In order to make sense of the nearly 500,000 keystrokes collected in Experiment 1, we developed a keystroke parsing algorithm that was instantiated in several data analysis programs. The goal of the parsing

Figure 4. **Parsing efficiency. Percentage of time attributed to the satisfaction of goals by the parsing algorithm.**

		Learning					Transfer		
Editor	Day	1	2	3	4	*M*	1	2	*M*
EMACS		94	94	95	95	95	93	95	94
ED		82	89	92	92	89	94	91	93
EDT		78	90	93	86	87	96	94	95
M		85	91	93	91	90	94	93	94

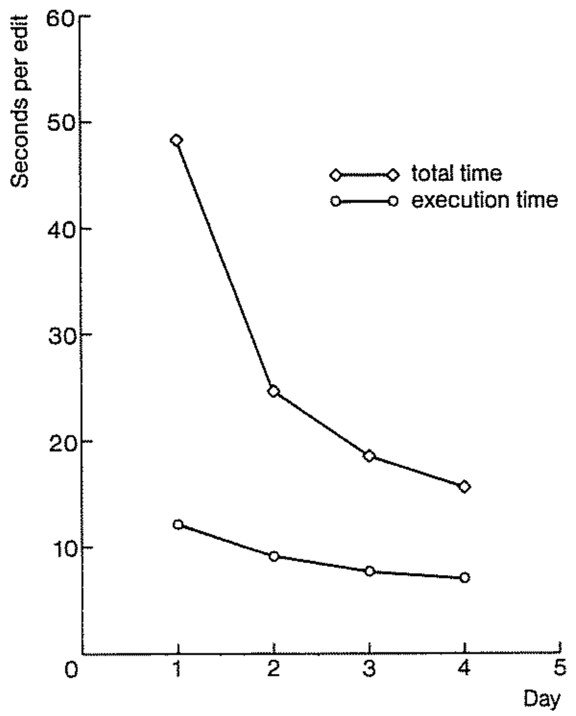
algorithm is to identify each burst and pause in the keystroke data and attribute it to the planning or execution of some text-editing operation. The general strategy is to simulate the editors by parsing commands and updating the contents of the file while collecting statistics on each interaction. A full description of our keystroke parsing procedure can be found in the Appendix. A short summary follows.

The parsing algorithm first segments the keystroke data from a single trial into six unit-task episodes corresponding to the six edits on the page. The algorithm further subsivides these six segments into their locate-line and modify-text components. The locate-line component (LL) includes not only the time and keystrokes spent moving to the site of the modification but also the time spent acquiring the unit-task from the manuscript (see Figure 3). The modify-text component (MT) is composed simply of the time and keystrokes spent modifying the text. Finally, these two components are split into planning and execution subcomponents. Operationally, we defined the execution subcomponent as the time from the first to last keystroke minus any interkeystroke pauses of greater than 2 sec. (As we shall see shortly, subsequent analyses provided corroboration for this cutoff point of 2 sec.) Likewise, we defined the planning component to be the sum of all pauses of greater than 2 sec. Typically, an LL or MT component begins with a long pause followed by keystroke bursts separated by short pauses.

Critical to the success of the parsing algorithm is that subjects move in an orderly fashion from unit-task to unit-task in the manuscript. Although the parser has certain methods for dealing with backtracking and skipped goals, in many cases the parser simply refuses to attribute pauses and keystrokes that fall outside of its expectations. Figure 4 shows one measure of the success of our parsing algorithm: the percentage of total time attributed to the satisfaction of goals. Generally, parsing efficiency ranges between 85% and 95%.

Overview of Microresults. The primary yield from our parsing analysis is a set of learning and transfer curves for the 12 methods corresponding to the 12 kinds of modifications in our editing task. Each of these curves is further subdivided into its LL and MT components, which are, in turn, split into planning and execution components. Given such a multilevel analysis, it should be possible to localize the learning and transfer effects observed at the macrolevel. We proceed with this microresults section by moving from the coarser to the finer levels of analysis, as just outlined. We start with a series of learning analyses and conclude with a series of transfer analyses.

Figure 5. **The aggregate method learning curve broken down into planning and execution components. Planning time is the difference between total time and execution time.**



First Learning Analysis: Relative Contributions of Planning Time and Execution Time

In this first analysis, we collapse across LL, MT, and the 12 methods and simply split total time per edit into its planning and execution components. One of the first things we discovered was that, apart from minor differences, all learning curves decomposed in this way have a characteristic shape, regardless of editor. Figure 5 shows this characteristic shape averaged across the various methods in all three editors. The two curves plotted are total time

Figure 6. **Correlations between dependent measures in the three editors in Experiment 1.**

Correlation		Editor		
		EMACS	*ED*	*EDT*
Execution time vs. keystrokes	LL	.97	.86	.98
	MT	.99	.97	.98
	M = .96			
Planning time vs. keystrokes	LL	.46	.83	.73
	MT	.59	.72	.65
	M = .66			

per edit and execution time per edit; planning time is merely the difference of these two.

Most striking is the fact that, whereas total time per edit drops from 48 sec on Day 1 to 16 sec on Day 4 (a 3 to 1 decrease), the execution time curve is relatively flat. Apparently, most of what a subject learns manifests itself as a reduction in planning time. Also, as we shall see shortly, the slight decrease in execution time can be attributed totally to a decrease in keystrokes rather than an increase in keying rate.

The ratio of planning time to execution time drops from 3.0 on Day 1 to 1.2 on Day 4. This means that subjects spend about 75% of their time planning on Day 1 and 54% on Day 4. This 54% figure compares favorably with the results of Card et al. (1983), who reported that the seasoned experts in their studies spend about 60% of their time planning. Our number is somewhat lower because a full third of the subjects used in our analysis were editing with *EMACS,* which requires substantially less planning time per keystroke than the line editors used here and also those used in Card et al. Specifically, the ratio of planning time to execution time on Day 4 for our *EMACS* subjects was .9, which is well below the line editor average of 1.3.

Having explored the general properties of method learning curves, we now move on to the next analysis, which again collapses across the 12 methods but partitions the keystroke data into two additional components: LL and MT.

Second Learning Analysis: Division Into LL/MT

Although the dramatic reductions occur in planning time, we have seen that modest reductions occur in execution time as well. The number of keystrokes per edit also decreases over the course of the experiment. The question naturally arises as to whether the reduction in execution time is due solely to the reduction in number of keystrokes. The first row of Figure 6 presents correlation data that establishes the connection between execution time and keystrokes in each of the editors. Separate data points were entered for each

of the 12 methods on 4 days of learning, making a total of 48 data points per correlation. (For this analysis, data was averaged across subjects.) As can be seen, the correlations are remarkably high ($M = .96$), giving strong evidence for an airtight linkage between these two variables. This lawful pattern of results attests to the soundness of our original decision to regard pauses of greater than 2 sec as a part of planning time. The fact that execution time can be computed by multiplying the number of keystrokes by typing speed indicates that our parsing procedure successfully segmented the planning and execution subcomponents.

To remove any doubt whatsoever concerning the exclusive relationship between reductions in execution time and keystrokes, we considered a plausible alternative hypothesis: that subjects were becoming better typists and were keying faster over the course of the experiment. To test this hypothesis, we performed multiple trials ANOVAs for the LL and MT components of the three editors using execution time per keystroke as the measure of keying rate. As expected, there was no significant main effect for days, which meant that subjects were keying at the same rate throughout the experiment—a sensible result given they were already skilled typists. Subjects averaged .47 sec per keystroke, or approximately 2 keystrokes per sec; there were no significant differences between editors. We conclude, then, that the speedup observed in the execution subcomponent is solely a result of subjects using fewer keystrokes to perform edits and not a result of faster typing.

Having isolated the source of speedup in execution time, we must now consider what causes subjects to use fewer keystrokes. Clearly, there are at least two possibilities: First, subjects' methods may be becoming more efficient because of either the acquisition of new operators or the more judicious use of existing operators. Second, subjects may be making fewer errors requiring fewer corrections. Fortunately, we can discriminate between these two possibilities with two new dependent measures. If subjects are acquiring more efficient methods, the number of keystrokes per command should decline over the course of the experiment. Likewise, if subjects are executing fewer errorful commands requiring fewer resubmissions, the number of commands per edit should decline.[1]

Figure 7 shows the results of 12 multiple trials ANOVAs performed on our two dependent measures for the MT and LL components of the three editors.

[1] The concept of command is clearly defined in the line editors to be a string of characters composed on a single line and terminated by a carriage return. However, the concept is not so clear in a screen editor like *EMACS*. For this analysis, we defined a command in *EMACS* to be any uninterrupted string of LL or MT keystrokes. For example, in the parse in Figure 20 (see Appendix), the LL component is composed of three commands. The first is the initial string of LL keystrokes, the second is the alphanumeric *f* followed by its deletion, and the third is the concluding string of LL keystrokes. This definition allows for our dependent measures, keystrokes per command and commands per edit, to have common referents in all three editors.

Figure 7. **Summary of ANOVA results for keystrokes per command and commands per edit. An x denotes a significant reduction in the marked component over the course of Experiment 1.**

		Editor		
Dependent Measure		*EMACS*	*ED*	*EDT*
Keystrokes per command	LL		x	
	MT			
Commands per edit	LL		x	x
	MT	x	x	x

As can be seen, five of the six commands per edit ANOVAs yielded a main effect for days, but only one of the six keystrokes per command ANOVAs did. This pattern of results strongly suggests that most of the reduction in keystrokes is due to fewer error episodes and not more efficient methods.

Given our success at reducing execution time to number of keystrokes, it is tempting to propose that planning time can be similarly reduced. If this were possible, a very simple model of learning and performance would emerge that associates some fixed amount of overhead with each keystroke and explains all speedup in terms of reductions in keystrokes. However, the results strongly suggest that the estimation of planning time is a much more complex undertaking. First, correlations between planning time and number of keystrokes are too low. The second row of Figure 6 shows that, whereas number of keystrokes predicted over 90% of the variance in execution time, it predicts only 40% of the variance in planning time. Of course, because both number of keystrokes and planning time are decreased, we would expect some correlation. Second, we performed six repeated-measures ANOVAs using planning time per keystroke as the dependent variable and found that both MT and LL components in all three editors were significantly decreasing on this measure, all $F(3, 9) > 5.0$, all $p < .05$. Also, significant main effects and interactions were found for the various unit tasks. This is additional evidence that planning time per keystroke is not a constant in our experiment.

Third Learning Analysis: Division Into Methods

So far in this microanalysis we have been concerned primarily with the acquisition of four components: LL and MT planning times and LL and MT execution times. We now increase the complexity of the analysis by a factor of 12 by decomposing the LL and MT components into their 12 constituent methods. Recall that these methods correspond to the 12 kinds of edits found in the manuscript and are derived by crossing the editing operations insert, replace, and delete by the data objects character, word, string, and line. The purpose of this analysis is to gain a better understanding of the microstructure of text editing and, therefore, set the stage for microanalyses of transfer.

As it is difficult to grasp the independent contributions of the dozens of learning components separately, we propose a simple model to summarize the results. This model attempts to account for the LL and MT planning time and LL and MT execution times for 9 of the 12 types of edits over the course of the experiment. We have excluded those edits concerned with manipulating lines (i.e., insert line, replace line, delete line) from the analysis for reasons to be discussed shortly. Our model is characterized by the following features:

1. All execution times are based solely on the number of keystrokes required to perform edits.
2. LL planning times vary only as a function of days of practice.
3. MT planning times vary not only as a function of days of practice but also as a function of editing operation (insert vs. replace vs. delete) and data complexity (character vs. word vs. string).

Our strategy for supporting the model is as follows:

1. Use correlations to show that execution time is strictly dependent on the number of keystrokes.
2. Use ANOVAs to show that LL planning time decreases over days but is independent of edit type.
3. Use ANOVAs to show that MT planning time decreases over days but also varies as a function of edit type. The model predicts main effects for both editing operation and data complexity but no interaction.

As for these three lines of evidence, the first has already been established. Figure 6 presented a series of correlations that demonstrated the strong link between number of keystrokes and execution time. Subsequent analyses confirmed the exclusive relationship between these two variables. As for the second, 3 × 3 × 4 repeated measures ANOVAs using LL planning time as the dependent measure confirmed that, in all three editors, LL planning time varied only as a function of practice, all $F(3, 9) > 12.08$, all $p < .001$, and not as a function of edit type.

As for the third line of evidence, that concerning MT planning time, the pattern of results is slightly more complicated. Figure 8 presents matrices of means for the nine methods in the line editors and *EMACS* averaged over Days 2 to 6. Because of high variability, data from the first day of learning have been excluded from this analysis. Data were collapsed across the line editors after an ANOVA comparing *ED* and *EDT* in terms of MT planning time yielded neither a main effect nor an interaction for editor. This, again,

Figure 8. **Matrices of planning time means for the line editors and *EMACS*.**

	Line Editors				*EMACS*			
	Insert	Replace	Delete	*M*	Insert	Replace	Delete	*M*
Char	11.8	13.0	11.3	12.0	4.2	4.8	2.9	4.0
Word	16.0	15.1	13.0	14.7	3.2	6.8	4.4	4.8
String	21.6	15.7	14.9	17.4	6.2	7.8	4.7	6.2
M	16.4	14.6	13.1	14.7	4.5	6.5	4.0	5.0

confirms our hypothesis that line editor MT components are virtually identical.

In the figure, we are first reminded of the gross disparity between line editor and *EMACS* MT planning times, which was mentioned in the first learning analysis. As for the predictions of the model, two 3 × 3 repeated measures ANOVAs yielded main effects for editing operation and data object in both the line editors and *EMACS*. However, the *EMACS* ANOVA also yielded an interaction—a result inconsistent with the predictions of our simple model but apparently an anomoly due to an unnaturally low entry for the *insert word* cell.

Line Operations. We have excluded the line operations from our simple model because they necessarily disrupt the main effects found for editing operation and data complexity by introducing a strong interaction. This interaction has two sources: First, in all three editors, the predominant method for *delete line* requires just two keystrokes (d followed by [CR] in the line editors and ↑k↑k in *EMACS*). Therefore, the *delete line* operation was much lower in both MT planning time and keystrokes than our model predicted. Second, *insert line* takes more planning time than *replace line* in *EMACS*, which is inconsistent with the general pattern. This reversal is most likely due to the difficulty subjects had "making space" for the line to be inserted, a problem well-documented by Mack, Lewis, and Carroll (1983).

This concludes the learning analyses from Experiment 1. We now move on to transfer analyses.

First Transfer Analysis: LL/MT

As noted at the beginning of the results section, we observed at the macrolevel near total transfer between the line editors and moderate transfer from the line editors to *EMACS*. We now ask whether it is possible to localize any of these macroeffects to either the LL or MT component. Besides shedding light on the sources of transfer, this analysis provides our first opportunity to study the intimate relation between learning and transfer in text editing.

At the end of this article is a section describing in explicit detail the simulation models used to make precise quantitative predictions of transfer.

However, at this point, we are concerned primarily with the results of empirical rather than formal analyses. In order to guide our understanding of the results, we will make rough predictions based on sketchy descriptions of the underlying models. Later, once the basic outlines of the phenomenon are understood, we will compare the observed results with quantitative predictions.

We now compare the editors in terms of both LL and MT components. All three editors use different methods to locate lines. However, the LL component spans not only locate-line procedures but also acquire-unit-task procedures and the uppermost nodes of the text-editing goal tree (see Figure 3). Because traversing the top nodes in the goal tree and encoding the edits are the same regardless of editor, we would expect moderate and roughly equal degrees of transfer among the three editors in the LL component based on these rules. However, the line editors share several additional rules pertaining to the selection of LL method and also specification of the secondary, carriage-return method. Therefore, LL transfer in the line editors should be somewhat higher than that between the line editors and *EMACS*.

The degree of similarity in the line editors is even greater in the MT component. Although the surface features of the MT commands in the two line editors are largely different, their underlying conceptual structures are nearly identical. This means that the line editor MT procedures share many high-level and intermediate-level nodes in their goal trees. For example, to replace a line in *ED*, one moves to the line to be replaced and types c for change. In *EDT*, one moves to the line to be replaced and types r for replace. In both editors, one then types in the new line. To exit the insert mode in *ED*, one types a period by itself on a line immediately followed by a carriage return. To exit the insert mode in *EDT*, one presses ↑z. Although these methods are quite different in surface symbols typed, they have the same underlying logical structure. This common structure is a likely source of positive transfer in the MT component.

Although line editor MT procedures are quite similar, line editor and *EMACS* MT procedures are largely different. However, they do share several rules for generating the top MT nodes in the goal tree and for inserting text. Therefore, we would predict nearly total transfer between the line editors in the MT component but substantially less from the line editors to *EMACS*.

Transfer Between Line Editors. Figure 9(a) presents line editor learning and transfer data for LL and MT planning time. We focus on planning time because it is not affected by such factors as typing speed and is, therefore, our purest measure of higher-level cognitive processing. Data is averaged across the first 2 days of learning and transfer for additional reliability.[2]

[2] As Card et al. pointed out, variability generally increases as the grain size of an analysis gets finer. To combat this effect, we collapse across days.

Figure 9. **LL/MT planning time transfer. Raw scores report the average number of seconds to either LL or MT.**

Between	Locate Line		Modify Text	
Line Editors	To *ED*	To *EDT*	To *ED*	To *EDT*
Transfer(3)	10.0	17.4	13.1	14.8
Learning(1)	19.0	44.7	45.9	53.3
Learning(3)	9.2	13.5	12.8	16.5
Transfer score (from Equation 1)	91%	87%	99%	105%

	LL		MT	
To EMACS	From Line Editors	From Typing	From Line Editors	From Typing
Transfer(5)	14.0	19.6	12.9	20.7
Learning(1)	27.0	27.0	2.76	27.6
Learning(5)	5.7	5.7	4.0	4.0
Transfer score (from Equation 1)	61%	35%	62%	29%

To characterize the magnitude of transfer, we use a transfer score introduced by Katona (1940) that measures the savings on a transfer task relative to a theoretical upper limit derived from learning data. This formula is insensitive to the amount of training prior to transfer, because the transfer savings (the numerator) is modulated by the degree of learning (the denominator). If subjects transfer to a new editor on day n of the experiment, the percentage of transfer is given by the following equation:

$$\%_{\text{transfer}} = \frac{M_{\text{lrn}}(1) - M_{\text{tran}}(\text{n})}{M_{\text{lrn}}(1) - M_{\text{lrn}}(\text{n})} \times 100 \qquad (1)$$

As an example, we will show how this equation applies to gauge transfer from *ED* to *EDT*. The denominator, which expresses the savings due to learning, is simply the difference between the means for Days 1 and 3 of the group that spends 4 days learning *EDT*. (Because subjects transfer from *ED* to *EDT* on the third day of the experiment, we use learning data from Day 3 to establish the theoretical upper limit of transfer.) The numerator, which expresses the transfer savings, is the difference between the learning mean on Day 1 and the transfer mean on Day 3. We see that this measure varies sensibly from 0 to 1, although negative values and values greater than 1 are possible. The former represents negative transfer, and the latter a kind of supertransfer one might observe in part-to-whole training.

The major result shown in Figure 9(a) is that the MT component exhibits more transfer than the LL component, with means of 102% and 89%, respectively. This is what an identical elements model would predict, given the line editor MT procedures have more in common than the LL procedures. Although MT transfer is slightly larger, LL transfer is still quite substantial.

We have mentioned that both line editors have as a secondary method the use of the carriage return to move forward one line in the file. It may be that if this method is in fact a source of transfer, transfer subjects would use it more frequently than control subjects. To test this hypothesis, we characterized the LL methods used in the line editors as either primary—that is, line addressing (10p) in *ED* and string searching (t 'unique') in *EDT*—or secondary—that is, the carriage return method just mentioned. It turns out that the use of the secondary method is more common in the transfer subjects (28% vs. 17%), although a two-way ANOVA confirming the difference was only marginally significant, $F(1, 12) = 2.1$, $p < .2$. It appears, however, that subjects transferring to a new line editor rely more on a secondary but familiar procedure for locating lines. This result suggests a general rule that, when more than one method is available, known methods assume an unnaturally prominent role in transfer tasks.

It should be mentioned that another source of LL transfer from *ED* to *EDT* lies not in *ED*'s LL component, but rather its MT component. It turns out that the specification of unique search strings when locating lines in *EDT* is identical to the specification of unique replacement strings when substituting text in *ED*. Although these subprocedures serve quite different high-level goals, it is conceivable that the upper nodes rewrite to identical subgoals that involve the same subprocedures. In this way, the MT string specification subprocedure in *ED* can be transferred whole cloth to serve an LL goal in *EDT*.

Transfer to EMACS. Figure 9(b) summarizes LL and MT planning time transfer from the line editors and typing to *EMACS*. As with the line editor data, means have been averaged across the first 2 days of learning and transfer.

LL transfer from the line editors to *EMACS* is somewhat smaller than that between the line editors themselves, as predicted. We have already enumerated the additional LL transfer sites in the line editors, which would explain this discrepancy. The amount of LL transfer from typing is less than that from the line editors, but is still greater than zero. To understand planning time transfer from typing, recall that subjects in the typing control group were required to type the edits that were marked on the manuscript. This means that they had practice encoding the edits from the manuscript. However, they had no exposure to the text-editing goal tree, as they were typing continuous text and not locating lines and modifying text. This explains the difference in LL transfer between the line editor and typing control groups.

The average MT transfer from the line editors and typing control to

EMACS was 46%, definitely less than the 102% observed between *ED* and *EDT*. However, this still seems rather high given that *EMACS* MT procedures share very little with line editor MT procedures, and there is no text modification per se in the typing manipulation at all. Here, then, is an interesting challenge to our identical elements model. Possible sources of transfer include such low-level operations as glancing at the manuscript and using the proper keying action on control and escape keys, but our data are unable to provide support for such conjectures. An additional candidate is the encode edit component just proposed as a source of LL transfer. Our experimenters report that, unlike the experts observed by Card et al. (1983), our subjects often required more than one look at the manuscript to represent edits, especially those manipulating strings and lines. Often these additional looks occurred after the modification had been located, that is, the period attributed to MT planning. More efficient encoding procedures gained through experience with the line editors and the typing manipulation would thus contribute to MT transfer. Although this and the other proposed sources might contribute to transfer, as a group they seem insufficient to account for the magnitude of transfer we observed. We therefore suspend any firm conclusions at this time. A better understanding of this perplexing result must wait for the results of the next, more detailed empirical analysis.

Second Transfer Analysis: MT Methods

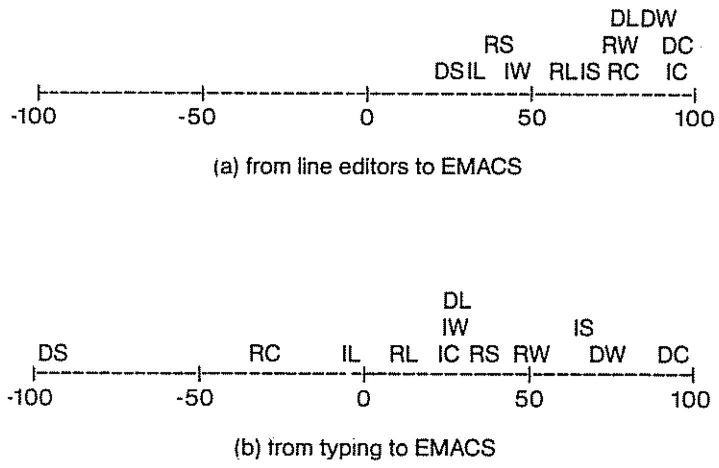
Our primary purpose in this section is to pursue the outstanding question concerning transfer from the LL/MT analysis: What is the source of the rather substantial transfer in the MT component from the line editors and typing to *EMACS*? Currently, the candidates include portions of the upper-level goal tree (good only for line editor subjects) and subskills for encoding the edit from the manuscript (good for both sets of subjects). In the present analysis, we can examine the differential transfer of all 12 MT methods and possibly localize the transfer of particular operators.

Figure 10 presents transfer scores expressing MT planning time transfer from the line editors and typing to *EMACS*. Once again, the scores are based on data from the first 2 days of learning and transfer. To facilitate interpretation, we have plotted the savings scores along number lines which show the relative orderings of the various subgoals.

Rather than attempt to understand the entire continuum of subgoal transfer data, we focus on those subgoals at either end of the distribution—those that exhibit the most and least transfer. Interestingly, the *delete character* subgoal shows the most transfer and *delete string* the least in both line editor and typing distributions. Another strong performer for both groups is *delete word*; *delete line* is strong for only line editor subjects.

What might explain such a pattern of results? Most compelling is the fact that these extreme sites are all instances of delete operations. Furthermore, in

Figure 10. **Number line showing ordering of unit tasks in terms of MT planning time transfer.**



both the line editors and the typing control, subjects do learn a deletion operator that could transfer to *EMACS*: the *delete* key. The *delete* key is used to edit malformed commands in the line editors and to correct typos in the typing control. (It is interesting to note that our task analyses based on error-free behavior failed to identify this area of commonality.) So, although the *delete* key is not used as part of any text-editing method that can be transferred whole cloth to *EMACS*, it is nevertheless available for use as the terminal node in the goal tree of some new method. This would explain not only the high level of positive transfer for *delete character* and *delete word* subgoals, but also the low-level and even negative transfer for *delete string*. There are much more efficient methods for deleting strings in *EMACS* than the repetitive use of the *delete* key.

An earlier analysis of the transfer of LL methods between the line editors suggested that, if many methods exist for accomplishing a goal and most are new but one is familiar, the familiar method will be favored. Applying this general rule to the present situation, if the *delete* key is a source of transfer to *EMACS*, line editor and typing control subjects should use the *delete* key more often in *EMACS* than *EMACS* control subjects. Examining the deletion methods used on the first day of learning and transfer, we found that, whereas *EMACS* control subjects used the *delete* key in 50% of their methods, line editor and typing control subjects used it in 68% of theirs. Although a chi-square test failed to confirm this difference as significant, this result does suggest that the *delete* key is being transferred.

The preceding discussion does not explain why the *delete line* subgoal shows a high-level of positive transfer in the line editor group but not the typing control. In *EMACS*, *delete line* is distinguished from other deletion operations by a specialized method so efficient it precludes the use of the *delete* key. This method has the same goal structure as the *delete line* method in the line editors but involves different keystrokes. In all three editors, one moves to the line requiring deletion and types the *delete line* operator. (In the line editors, the operator is d, and in *EMACS*, it is ↑k↑k.) There is no such goal structure in the typing control, hence the difference in *delete line* transfer.

Here, then, in the delete operations, are two additional sources of MT transfer to *EMACS*. The first, the *delete* key, contributes to either positive or negative transfer, depending on its interaction with unit-task. The second, the *delete line* goal structure, exists in only the line editor group, and thereby contributes to the difference between line editor and typing control transfer. These specific cases illustrate the general point that, although text modification procedures in the line editors and *EMACS* are quite different in terms of overall goal structure, the two do share a number of leaf components.

2.3. Summary of Experiment 1

We have just completed a rather detailed analysis of the data. It is worthwhile to stress some of the general conclusions supported:

1. Degree of transfer does seem to be a function of the overlap in number of elements.
   a. Information-processing models like GOMS help to define shared goal structures. For instance, the massive transfer in MT between the line editors occurred because of identical goal structures for different physical commands.
   b. Even in cases where tasks do not share goal structures, there can be considerable transfer at the level of leaf nodes.
2. A complex task like text editing can be decomposed into parts and each part seems to be learned separately.
   a. Components such as LL are performed and learned independently of the context in which they occur.
   b. Time to perform highly practiced components such as typing do not speed up, but the planning components show very rapid speed up.
   c. Various components of the task seem to decompose cleanly into planning and execution time.

In summary, although the task of text editing is complex, it appears that under the right analyses the underlying behavior is not. In line with Simon's (1968) long-standing claims, the complexity of the behavior *simply* reflects the complexity of the task.

Negative Transfer. As mentioned previously, the issue of negative transfer complicates simple identical elements models of transfer. In the microanalysis, however, we found evidence for only one instance of this effect: the inefficient use of the *delete* key in *EMACS* to satisfy the *delete string* subgoal. This is not a true instance of procedural interference as commonly conceived, however. Procedural interference is usually thought of as one procedure disrupting the successful execution of another by making it slower or more errorful. In this case involving the *delete* key, the deletion procedure is not slower or more errorful, but rather is executing quite successfully. The problem is simply that this familiar procedure brought over from the line editors and typing is less efficient than other novel procedures in *EMACS*. If anything, the *delete* key is interfering with the acquisition, not the execution, of other procedures. This phenomenon can be likened to the classic Einstellung effect (Luchins, 1942), where the flawless execution of a familiar procedure in a new situation similarly leads to nonoptimal performance. Therefore, this instance of negative transfer seems to fit in rather well with our identical elements model.

3. EXPERIMENT 2: IDENTICAL ELEMENTS AND NEGATIVE TRANSFER

The purpose of the second experiment is twofold: (a) we wish to verify and extend our multicomponent view of learning and transfer by making additional transfer predictions, and (b) we wish to press the issue of negative transfer to the limit. We do this by importing a classic interference paradigm from verbal learning research into the domain of text editing.

For these purposes, we had to create a new, special kind of editor for the transfer task. This editor is identical to *EMACS*, with the one exception that all the control and escape keys are replaced or scrambled. For example, whereas in regular *EMACS*, ↑d deletes a character and ↑e moves to the end of a line, in this new editor, ↑d moves down a line and ↑e deletes a word. In all, almost half of the commands in the new editor are bound to keys used for contrary purposes in *EMACS*, with the remainder bound to entirely new keys. We call this new editor *perverse EMACS*, for obvious reasons.

Were a subject to learn *EMACS* first and then *perverse EMACS*, one would reasonably expect a lot of negative transfer. In fact, if we regard the functions of the keys as the stimuli and the bindings as the responses, we have an approximation to the classic A-B, A-Br interference paradigm used in verbal learning research (Postman, 1971). However, the important observation is that an identical elements model based on a production system analysis predicts strong positive transfer in this situation. Using the standard goal-tree representation of skill, *perverse EMACS* is identical to *EMACS* except that the terminal nodes executing specific *EMACS* keystrokes have been replaced. In other words, *EMACS* and *perverse EMACS* share all the internal nodes of their goals trees and differ in only the leaves.

The basic plan of our experiment is to teach subjects *EMACS*, then *perverse EMACS*, and finally *EMACS* again. Our major prediction is that each of the transitions will be marked by strong positive transfer. Not only will subjects' initial training on *EMACS* transfer to *perverse EMACS*, but also their continued training on *perverse EMACS* will serve to improve their performance on *EMACS* when they return to it. Aside from these general predictions, our further claim is that transfer will be localized primarily to the various planning components in the editors, because the areas of overlap lie mainly in the underlying goal structures and not the surface procedures.

Our experiment has one other feature which serves as an additional source of transfer predictions. We compare the performance of one group that passes through the basic *EMACS, perverse EMACS, EMACS* cycle just outlined with another that passes though the same cycle except that it learns one of the line editors first. The crucial manipulation here is that we control the amount of practice the second group gets so that, with its combination of line editor and *EMACS* training, it is performing as well on *EMACS* prior to transfer to *perverse EMACS* as the first group that has *EMACS* training alone. Although the groups will perform equivalently on *EMACS*, we predict that they will show differential transfer to *perverse EMACS*. This is due to the fact that the two groups have different learning histories and, therefore, different compositions of text-editing skill. Our specific prediction is that the transfer task will favor the group with the combination of line editor and *EMACS* training.

This prediction is based on the following analysis: The group with prior line-editor experience achieves equal overall performance prior to transfer by having the components common to *EMACS* and the line editor better practiced and the components unique to *EMACS* less practiced. Presumably, any component general to a line editor and *EMACS* would transfer completely to *perverse EMACS*, but those components specific to *EMACS* would not. Thus, the line editor group shows more transfer reflecting a stronger general component.

3.1. Method

Subjects. Subjects were eight women from the same population as in Experiment 1.

Materials. Three editors were used in this experiment, *EMACS, perverse EMACS*, and *EDT*. The decision to use *EDT* instead of *ED* as the line editor was entirely arbitrary. The functionalities of *EMACS* and *EDT* were identical to those in the first experiment and are summarized in Figure 1. Figure 11 presents a full comparison of the commands in *EMACS* and *perverse EMACS*. In the context of regular *EMACS*, *perverse EMACS* is certainly a treacherous editor. However, every effort was made to make the commands sensible in their own right. In fact, we tried to preserve many of the more user-friendly design features of *EMACS*. For example, command names are mnemonic: ↑u moves the cursor up one line and ↑d moves it down. In addition, commands are grouped in pairs:]e deletes a character and ↑e a word. In this latter case, however, just to add to the confusion, the functionality of the escape and control keys within the pairs are reversed, so that the escape version of the command operates on characters and the control version operates on words. (Just the opposite is true in *EMACS*.) Finally, we designed *perverse EMACS* to approximate regular *EMACS* in its proportion of control to escape key commands and also left-handed to right-handed commands. Our purpose in all of this was to make the new version of *EMACS* as learnable as the old, so that differences in performance between the two might be attributed to something besides absolute differences in the editors.

Figure 11. **Screen editor command summary for Experiment 2. In the table, ↑ denotes a control character and] an escape character.**

Command Type	Action	*EMACS* Binding	*Perverse EMACS* Binding
Locative	forward character	↑f	]r
	forward word	]f	↑r
	backward character	↑b	]l
	backward word	]b	↑l
	beginning of line	↑a	↑f
	end of line	↑e	↑b
	previous line	↑p	↑u
	next line	↑n	↑d
Mutative	delete character	↑d	]e
	delete word	]d	↑e
	delete previous character	DELETE	↑a
	delete line	↑k	↑w

Design. Our primary task in designing this experiment is to give two groups different amounts of general and specific practice but equate their overall performance on *EMACS* prior to transfer. We calculated from Experiment 1 that a group with roughly 2 days of *EMACS* practice will perform as well as another with 2 days of *EDT* and 1 day of *EMACS* practice. More specifically, we calculated that 2 days of *EMACS* practice would leave subjects with slightly better performance and planned to stop the subjects in this group on Day 2 when their performance reached the criterion level. Here, then, is our method for matching the two groups.

The full design for the experiment is as follows. One group (hereafter called the *line-and-screen* group) starts with 2 days of *EDT* followed by 1 day of *EMACS*, 2 days of *perverse EMACS*, and 1 day of *EMACS*. The second group (hereafter called the *screen-only* group) starts with 2 days of *EMACS*, followed by 2 days of *perverse EMACS*, and 2 days of *EMACS*.

Procedure. The procedure was nearly identical to that used in the first experiment. Subjects spent 3 hr per day editing the standard manuscript (see Figure 2). With each new editor, subjects were given a $\frac{1}{2}$ hr introduction by the experimenter that included both a description of the commands and a demonstration of two trials. The only difference in the procedure was that, on Day 2 of the experiment, members of the screen-only group were stopped prematurely by the experimental program if their performance dipped below a target level. We determined this target level by yoking each subject from the screen-only group to another run earlier in the line-and-screen group. Pairings were based on rank orderings of subjects from performance on the first day of *EMACS*. As expected, all screen-only subjects attained their target levels on the second day of the experiment.

3.2. Results

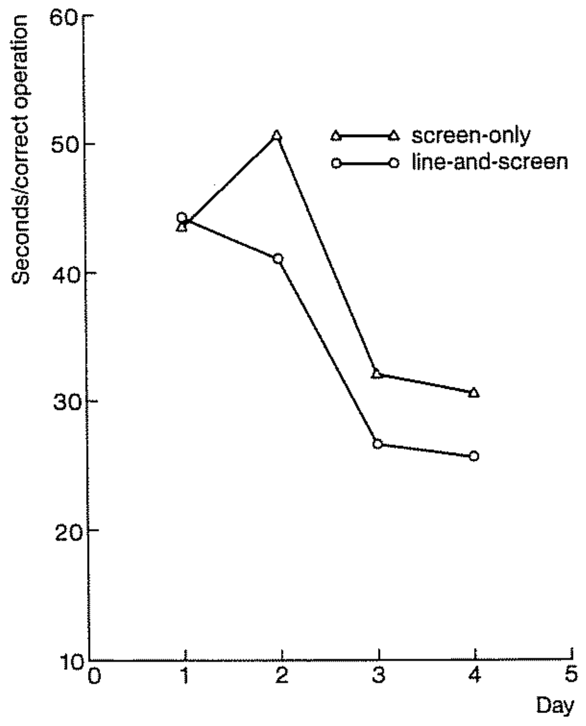
This results section has two major parts corresponding to the macro and micro levels of data analysis. The macroanalysis describes transfer in terms of aggregate measures like time per correct edit and keystrokes per trial. The microanalysis describes attempts at localizing transfer through the application of the parsing algorithm.

Macroanalysis of Transfer

Figure 12 presents data that confirms our prediction concerning differential transfer to *perverse EMACS*. Plotted is time per correct edit for the 2 days of *perverse EMACS* as well as 1 day before and after. (We have plotted Days 2 through 5 of the experiment for the screen-only group and Days 3 through 6 for the line-and-screen group. This aligns the *perverse EMACS* data to facilitate comparisons.) As shown, the two groups are matched quite well on *EMACS* prior to transfer (43.5 and 44.2 sec per edit for the screen-only and line-and-screen groups, respectively). However, these functionally equivalent groups are separated by transfer to *perverse EMACS*. A one-tailed matched-groups t-test confirms that the average performance on *perverse EMACS* of the screen-only group is significantly worse than that of the line-and-screen group, $t(3) = 2.3$, $p < .05$. Interestingly, the last day plotted, which represents the return to *EMCS*, shows that the two groups never quite recover their former equivalance: The screen-only group continues to lag behind.

As might be expected given previous results (Singley & Anderson, 1985), the pattern observed in the timing data is duplicated almost perfectly in the keystroke data. Our matching procedure inadvertently matched groups on

Figure 12. **Transfer to *perverse EMACS* in terms of time per correct edit. The curves have been aligned so that the first day represents performance on *EMACS* prior to transfer, Days 2 and 3 performance on *perverse EMACS*, and Day 4 the return to *EMACS*.**



keystrokes as well as time per edit, giving further credence to the position that number of keystrokes is a major predictor of performance in text editing (Card et al., 1983). Once again, the two groups are separated by the transfer task, but this time the matched-groups t-test was only marginally significant, $t(3) = 1.7$, $p < .09$. If we accept the keystroke difference as real, however, it sheds an entirely new light on our interpretation of the results. We had predicted the screen-only group's inferior performance in terms of a weaker, less-practiced general component that runs more slowly in *perverse EMACS*. This kind of explanation is based on a strong identical elements model of transfer that makes no appeal to negative transfer. However, the disadvantage in keystrokes raises the possibility that the screen-only group's strong specific component is playing a role as well. The group's well-practiced *EMACS*-specific methods may be interfering with the acquisition and use of efficient methods in *perverse EMACS*. With this kind of explanation, the screen-only group is in a kind of double jeopardy: both its general and specific components are liabilities. The general component is weaker and contributes less to positive transfer, whereas the specific component is stronger and contributes more to negative transfer.

The further interpretation of the results of this experiment, as well as the status of identical elements models of transfer generally, seems to hinge on the issue of negative transfer. For this reason, we devote the remainder of the macroanalysis and the entire microanalysis to its exploration. We focus our attention on the screen-only group, because its poorer performance makes it a more likely victim.

Baseline Comparisons. To gain a better perspective on the absolute performance of the screen-only group on *perverse EMACS*, we compare the screen-only data to the *EMACS* learning curve from Experiment 1. Because as editors *EMCS* and *perverse EMACS* are practically equivalent on all objective measures, we can use the *EMACS* learning curve as a rough approximation to a *perverse EMACS* learning curve and thereby gauge whether the screen-only group experiences positive or negative transfer overall.

Figure 13 compares the screen-only and *EMACS*-control groups on time per correct edit over 6 days of editing. Recall that the screen-only group spent Days 3 and 4 of the experiment using *perverse EMACS*; all other data points in the figure represent performance on *EMACS*. We see that the two curves follow each other fairly closely. In fact, *t*-tests revealed that the two groups were not statistically different on 5 of the 6 days. Only on the third day, the first day of transfer to *perverse EMACS*, did the groups differ significantly, $t(6) = 2.9$, $p < .05$. These results highlight the overwhelming similarity and positive transfer between the two editors. First, compared to Day 1 using *EMACS*, there is large positive transfer for the screen-only group on Day 3 to *perverse EMACS*. The difference between the two groups on Day 3 represents the temporary deficit suffered by the screen-only group while learning the specific rules of *perverse EMACS*. By Day 4, this deficit has largely disappeared, and on Day 5, when the screen-only group returns to *EMACS*, it picks up at the same point on the learning curve as the group that had stayed with *EMACS* all along. Thus, although there is some reason to suspect that the screen-only group is experiencing negative transfer at some level, most measures at the macrolevel indicate massive positive transfer.

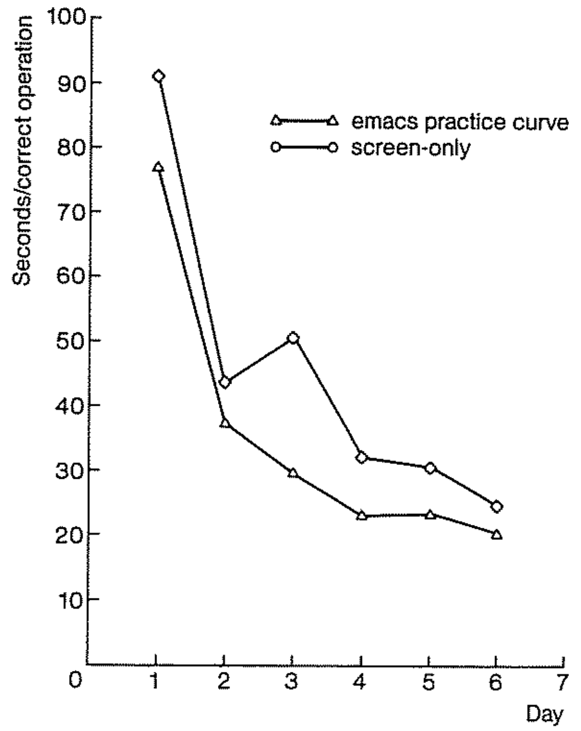
Microanalysis of Transfer

At finer grains of analysis, it should be possible to identify and characterize sources of negative transfer that might have been washed out at more aggregate levels. We continue to focus on the screen-only group.

First Transfer Analysis: LL/MT

We have pointed out previously that in using the standard goal-tree formulation *EMACS* and *perverse EMACS* differ in only their terminal nodes.

***Figure 13.* Comparison of screen-only group from Experiment 2 with the *EMACS* control group from Experiment 1 in terms of time per correct edit.**



In other words, both MT and LL planning components overlap completely, but the execution components are largely different. It turns out that, whereas the LL execution component is completely different in the two editors, the MT execution component overlaps somewhat because insert operations make use of the standard alphanumeric keys that have not been rebound. Given this analysis, we would expect large and equal amounts of transfer in the LL and MT planning components, less transfer in the MT execution component, and essentially none in the LL execution component. Of course, if execution components are somehow interfering with one another, transfer could be negative.

To measure LL/MT transfer, we have devised a new transfer score that is similar in spirit to the score used in Experiment 1. The goal once again is to measure transfer relative to the amount of learning that takes place with the same amount of practice. Because we did not make an independent assessment of learning in *perverse EMACS*, we use regular *EMACS* learning data from Experiment 1 to estimate learning rates. This approximation is quite

Figure 14. **LL/MT transfer from *EMACS* to *perverse EMACS*.**

		LL	MT
Planning time	Transfer(3)	16.0	17.7
	Learning(1)	29.4	34.8
	r	.67	.78
	Transfer score (from Equation 2)	68%	63%
Number of keystrokes	Transfer(3)	10.7	20.8
	Learning(1)	8.7	18.8
	r	.10	.17
	Transfer score (from Equation 2)	−234%	−61%

good given that the two editors are functionally and structurally equivalent. The new transfer score is given by the following equation:

$$\%_{\text{transfer}} = \frac{M_{\text{lrn}}(1) - M_{\text{tran}}(n)}{M_{\text{lrn}}(1) * r} \times 100 \qquad (2)$$

As in the earlier equation, the numerator represents the speedup due to transfer and the denominator the speedup due to learning. The Day 1 learning mean ($M_{\text{lrn}}(1)$) is taken from the screen-only group's performance on the first day of the experiment. To calculate the speedup due to learning, we multiply this mean by a fraction, *r*, which represents the proportion of speedup observed in Experiment 1 for the component in question. In this way, the learning baseline is supplied by the screen-only group, but learning rates are supplied by Experiment 1. This allows for a more sensitive within-subjects measure of transfer that behaves identically to the between-subjects measure used in Experiment 1.

Figure 14 presents LL/MT transfer scores for both planning time and keystrokes. We see that our predictions concerning LL/MT transfer are largely confirmed. Planning time transfer is substantial ($M = 66\%$) and virtually identical for LL and MT components. Keystroke transfer (i.e., execution component transfer) is much less than planning time transfer, and LL is much worse than MT, as predicted. Most striking is that the keystroke transfer is not small or zero but negative. On average, subjects strike an extra two keystrokes per edit on the first day of *perverse EMACS* compared to the first day of *EMACS* on both LL and MT components. Thus, it appears that *EMACS*-specific methods are somehow interfering with the acquisition of new methods in *perverse EMACS*. The exact nature of this interference remains to be seen.

Second Transfer Analysis: Division Into Methods

Although transfer at the macrolevel was overwhelmingly positive, we have now identified two sources of negative transfer in the LL and MT execution components. In this analysis, we attempt to further localize the sites of negative transfer by examining individual unit tasks. The analysis is restricted to the MT component because LL procedures are essentially the same for all unit tasks and should not differ in terms of transfer. In addition, the three unit tasks involving line operations are excluded because MT procedures for these three share nothing with the others and add unnecessary complexity to the analysis.

As stated previously, *insert* operations are identical in the two editors, but *delete* operations are grievously different. This means that, although the transfer observed in the LL/MT analysis for the MT execution component was negative overall, that subset of unit tasks involving insertion should show substantial positive transfer. Negative transfer should be restricted to those unit tasks that involve deletion. Because replacement involves both insertion and deletion, transfer scores for those unit tasks should lie somewhere between the other two.

To further dramatize the fact that positive transfer dominates the planning components and negative transfer is restricted to a subset of execution components, we first examined MT planning time for each of the nine unit tasks. As expected, a 3 × 3 ANOVA using the transfer score in Equation 3 as the dependent measure yielded no main effects or interactions, thereby confirming that MT planning time transfer for all nine unit tasks was equally positive.

When we performed the same ANOVA using keystroke transfer as the dependent measure, a main effect for editing operation emerged, $F(2, 12) = 123.6$, $p < .001$. As predicted, insertion operations exhibited substantial positive transfer ($M = 110\%$) and deletion operations massive negative transfer ($M = -240\%$). Finally, replace operations were in the middle ($M = -100\%$). Here, then, is strong support for the view that deletion operations are the source of negative transfer in the MT execution component.

Third Transfer Analysis: MT Methods

We now know that the acquisition of *EMACS* deletion methods somehow interferes with the acquisition of *perverse EMACS* deletion methods. Critical to the status of our identical elements model of transfer is the nature of this interference. In Experiment 1, the one case of negative transfer was characterized in terms of the positive transfer of a nonoptimal method. No real evidence was found for procedural interference in the classic sense, and a fairly strong version of our identical elements model was retained.

The pressing question now is whether the rather substantial negative

Figure 15. **Distribtion of operators in deletion unit tasks for (a) *EMACS* and (b) *perverse EMACS*.**

	EMACS			*Perverse EMACS*		
Operator Unit Task	del-char (↑d)	del-pre-char (DELETE)	del-word (]d)	del-char (]e)	del-pre-char (↑a)	del-word (↑e)
delete character	78	16	6	97	0	3
delete word	66	22	12	81	3	16
delete string	60	10	30	75	0	25

transfer observed in this experiment can be similarly explained. To answer this question, we performed a qualitative analysis of the methods used for the deletion of characters,words, and strings in both *EMACS* and *perverse EMACS* by screen-only subjects on the first 4 days of the experiment. As can be seen in Figure 11, there are four deletion operators in each editor: delete character marked by cursor (*del-char*), delete word marked by cursor (*del-word*), delete character to left of cursor (*del-pre-char*), and delete from current cursor position to the end of the line. This last operator will be ignored because it was not used in any of the three unit tasks in either editor. Figure 15 shows the distribution of the remaining three operators in the deletion unit tasks of both editors.

We first see that in *EMACS* the operator of overwhelming preference is *del-char* (↑d). Subjects adjust their operator selections somewhat by the amount of text to be deleted, as evidenced by the increased use of *del-word* (]d) on words and strings. However, even in the *delete string* unit task, ↑d enjoys a 2 to 1 advantage over the more optimal]d. Looking at the *perverse EMACS* results, we see that *del-char* (now bound to]e) is again the runaway favorite, but now its dominance is even greater. The *del-pre-char* operator, which played a minor role in *EMACS*, has all but disappeared in *perverse EMACS*. Interestingly, of the three deletion operators used in *perverse EMACS*, only one,]e, did not have a previous binding in *EMACS*.

Thus, it appears that negative transfer can once again be explained in terms of the positive transfer of a nonoptimal method. Transfer is negative because, whereas the use of *del-char* in *EMACS* involved a single keystroke, its use in *perverse EMACS* involves two. (The escape key in the]e operator must be struck independently and, therefore, counts as an extra keystroke.) Therefore, what was a nonoptimal method in *EMACS* becomes even worse in *perverse EMACS*. In addition, the use of this nonoptimal method may be amplified somewhat by a kind of fan effect (Anderson, 1983), which penalizes those operators with multiple bindings and enhances those with single bindings. In our case, the operator with a single binding just happened to be *del-char*, which is already dominant and, as we've seen, very inefficient. However, even this

secondary effect is probably not true procedural interference, in that its roots are most likely in the declarative representation of the skill. Factors like fan probably influence the initial operator selections made during the early interpretive phase of skill acquisition. Such influences can certainly affect the course of knowledge compilation and ultimately the compiled representation of the skill.

3.3. Summary of Experiment 2

The results of our second experiment largely reinforce the conclusions of the first. Text editing does seem to be a skill that can be hierarchically decomposed into independent subcomponents. Positive transfer is to be understood in terms of the number of shared components among skills. In addition, the second experiment has shown that negative transfer has a similar explanation. Negative transfer occurs when the use of shared components in a new skill is suboptimal. Again we see that the transfer of text-editing skill can be understood in terms of an identical elements model when that model is defined by an appropriate cognitive representation.

4. SIMULATION MODELS AND QUANTITATIVE PREDICTIONS

Up to now, we have been content to make transfer predictions based on rather qualitative analyses of similarity. We now pool data from the two experiments and make quantitative predictions of transfer based on detailed task analyses of text-editing skill in the four editors. The product of our task analyses is a set of production system models which simulate skilled, error-free text-editing behavior in each of the editors. As mentioned previously, the underlying goal structures used in our models are based largely on the GOMS keystroke level analysis of Card et al. (1983, pp. 165–166). The GOMS model uses a strict hierarchical control structure to model expert, error-free text-editing behavior. With such a restricted goal structure, the GOMS model is not a true instance of a production system, although it can be easily adapted to one. To recast the GOMS model as a production system, we assume that several productions fire to create the top-level goal structure in Figure 3. In response to each of these major goals, additional productions fire to create subgoals and eventually actions. This production system analysis is essentially identical to the GOMS formulation. In our simulations, we use the GRAPES production system language (Sauers & Farrell, 1982), which supports the construction of hierarchical goal trees and restricts production firings to those relevant to the current goal.

When using such production system models, a first approximation to a

Figure 16. Listing of productions used in the text-editing simulations. Productions are categorized in terms of goal (LL vs. MT) and range of application. Production frequency refers to the number of times a production fires every two trials.

Component	Category	Number of Rules	Total Frequency
LL	GENERAL	6	96.0
	LINE	10	55.4
	ED	5	49.7
	EDT	7	73.4
	SCREEN	13	89.0
	EMACS	7	62.0
	PERVERSE EMACS	7	62.0
MT	GENERAL	3	32.0
	LINE	24	105.3
	ED	6	23.0
	EDT	5	14.0
	SCREEN	18	70.0
	EMACS	4	18.0
	PERVERSE EMACS	4	18.0

transfer prediction would involve comparing two sets of productions for different editors. To the extent that the production sets overlap, transfer would be positive from one skill to the other. To get a somewhat more accurate prediction, we assign weights to the productions according to their frequency of use in the transfer task. A production that fires frequently in the transfer task contributes more to the time estimate than one that fires seldomly. This point in fact figures prominently because the productions that generate the upper-level goal structures common to all editors are relatively high-frequency productions, firing in service of every unit task.

A total of 118 distinct production rules are used to simulate behavior in the four editors. A good proportion of these rules do double-duty, that is, they apply in more than one editor. Figure 16 summarizes the rules and categorizes them in terms of whether they contribute to the LL component or the MT component.[3] Furthermore, these rules are categorized according to their range of application. The categories are as follows:

- *GENERAL*. Rule applies in all four editors.

[3] As alluded to previously, some productions that do not strictly fire in service of the LL goal still contribute to the LL component because they add to the initial long pause before the first LL keystroke. These include rules for generating the upper portion of the goal tree and rules for acquiring the unit task from the manuscript.

- *LINE*. Rule applies in both line editors: *ED* and *EDT*.
- *SCREEN*. Rule applies in both screen editors: *EMACS* and *perverse EMACS*.
- *ED*. Rule is specific to *ED*.
- *EDT*. Rule is specific to *EDT*.
- *EMACS*. Rule is specific to *EMACS*.
- *PERVERSE*. Rule is specific to *perverse EMACS*.

Note that there are no rules that are common to a specific line and screen editor (e.g., *EDT* and *EMACS*).

Also shown in Figure 16 is an estimate for total frequency of occurrence for each set of rules. These estimates were derived by simulating each production set on 10 randomly selected trials from the experimental manuscript. The numbers reflect the average number of firings of a rule on any two trials that contain the 12 kinds of edits our subjects had to perform. Thus, for a rule that fires once on every unit task (e.g., the rule that sets the subgoals of LL and MT) the frequency of occurrence is 12. The numbers in the figure are simply summations over all rules in the set.

There are four rules common to all editors that are not represented in the figure and make no contribution to our calculations. Two of these are rules for typing, and it has been shown in the learning analyses that there is virtually no speedup in the execution (i.e., typing) component. It is a general principle that a component which exhibits no learning can have no impact on transfer. Therefore, we exclude typing productions from the analysis. The productions shown in the figure contribute solely to the various planning components, so we restrict ourselves to predictions of planning time in these transfer analyses.

The other two excluded rules concern failing to acquire the next unit task and terminating with success when the acquire-unit-task goal fails. These rules fire in succession at the very end of each trial, and the associated pause is not included in any of our measurements.

Because it is impossible given space limitations to describe each rule in detail, we give brief descriptions of each category plus one or two examples. Those interested in a fuller understanding of the simulations are encouraged to write us for the code. Here, then, are summaries of each category:

LL-GENERAL. This contains rules for generating the upper levels of the goal tree in Figure 3 and rules for acquiring edits from the manuscript. Here is a version of one rule:

```
PRODUCTION execute-unit-task-G

IF the goal is to execute-unit-task
THEN set as subgoals to
  1) locate line
  2) modify text.
```

As mentioned previously, these are all relatively high-frequency rules and, therefore, make strong contributions to LL transfer among the editors.

LL-LINE. This contains rules for deciding between primary and secondary LL methods in the line editors, terminating commands with carriage return (this rule also appears in MT-line), recognizing when movement is required and when it's not, and rules for determining whether a string search is successful and how to pad the string to make it unique. It is interesting to note that these latter rules only apply in *EDT* because *ED* uses number rather than string addressing methods for locating lines. However, these rules appear in the MT component of *ED* because unique strings must often be selected as arguments to the substitution command. Therefore, these rules appear in the LL-line category rather than LL-*EDT*.

Here is a version of a rule from this set:

```
PRODUCTION choose-ll-secondary-LINE

IF the goal is to choose a command
  and the supergoal is to locate line
  and the current line is =line1
  and the target line is =line2
  and =line2 is immediately after =line1
THEN use the secondary carriage return method.
```

This rule implements the decision rule to select carriage return to move down a single line. This secondary method, of course, is common to both line editors.

LL-ED. This contains rules that specify the primary LL method in *ED* and rules that count lines of the manuscript in order to supply the line number argument. Here is the rule that specifies the method:

```
PRODUCTION ll-primary-method-ED

IF the goal is to enter a command
  and the command is LL-primary
THEN set as subgoals to
  1) specify the line number
  2) specify the command symbol.
```

This rule ultimately leads to the generation of a command such as 10p, which positions the user on the tenth line of the file.

LL-EDT. This contains rules that specify the primary LL method in EDT and rules for iterating through lines of the manuscript to test the uniqueness of a search string. This is the rule that specifies the primary LL method in *EDT*:

```
PRODUCTION ll-primary-method-EDT

IF the goal is to enter a command
  and the command is LL-primary
THEN set as subgoals to
  1) specify the command symbol
  2) specify the search string delimiter
  3) specify the search string
  4) specify the search string delimiter.
```

This rule leads to the generation of a command such as t 'hello', which positions the user on the first line containing the string 'hello' following the current line.

LL-SCREEN. The LL methods in screen editors like *EMACS* and *perverse EMACS* are much more precise than those in the line editors because both horizontal and vertical positions are specified. The LL rules common to both screen editors include rules for choosing among the various LL operators (e.g., forward-word, backward-word, beginning-of-line) and special-case rules for stopping in position depending on the direction of movement. For example, the user stops immediately to the right of certain modifications when coming from the right, but immediately to the left when coming from the left. Two distinct rules are required to model this behavior.

Here is the rule that chooses to apply to the forward-word operator:

```
PRODUCTION choose-forward-word-SCREEN

IF the goal is to move horizontally on a line
  AND the cursor is to the left of the modification
  AND one or more words separate the cursor and the
    site of the modification
THEN choose forward-word.
```

A separate rule retrieves the command symbol associated with the forward-word operator. These rules, however, are different in the two screen editors, because forward-word is]f in *EMACS* and ↑r in *perverse EMACS*.

LL-EMACS. These rules simply retrieve the bindings associated with the various LL operators in *EMACS*. For example, this rule retrieves the binding for next-line:

```
PRODUCTION next-line-EMACS

IF the goal is to specify the command symbol
  AND the command is next-line
THEN set as a subgoal to type ↑n.
```

LL-PERVERSE. The rules in this set are completely analogous to those in LL-*EMACS*. The LL operators simply have different bindings in *perverse EMACS*.

MT-GENERAL. This contains rules for setting the upper-level goal structure for modify-text, inserting text within a particular method, and verifying the location of an edit. This last operation is technically part of LL but is counted as part of MT by our parsing algorithm.

This rule sets the goal structure for modify-text:

```
PRODUCTION modify-text-G

IF the goal is to modify-text
THEN set as subgoals to
  1) choose a method
  2) use the chosen method
  3) verify the edit.
```

MT-LINE. The line editors share many rules for MT. Some of these are rules for selecting a particular MT operator; others concern the specification of string arguments to the heavily used substitution command. In this latter category, many deal with the management of space. For example, one rule states that if the goal is to delete text that has space on both sides, then one of those spaces should also be deleted. Similar considerations come into play for text insertion. For example, this rule deals with the insertion of a word or string of words into a line of text:

```
PRODUCTION second-arg-sub-insert-middle-space-LINE

IF the goal is to specify the second argument to
  the substitution command
  AND the modification is the insertion of a word or string of words
THEN pad the insertion with a space on the end.
```

This rule insures that all words are separated by spaces following text insertion.

MT-ED. These rules retrieve the particular MT command bindings for *ED*. For example, this rule retrieves the binding for replace-line:

```
PRODUCTION replace-line-ED

IF the goal is to specify the command symbol
  AND the command is replace-line
THEN set as a subgoal to type r.
```

MT-EDT. These rules supply the command bindings for *EDT* and are almost completely analogous to those in *MT-ED*. One difference is that the rule for supplying the syntactic terminator for the substitution command in *ED* is missing in *EDT* because no syntactic terminator is required (see Figure 1).

MT-SCREEN. MT rules shared by the screen editors concern the selection of MT methods and also the management of space, which, as just discussed, is also a source of common rules in the line editors. As an example of the former type, this rule selects the kill-line deletion method:

```
PRODUCTION choose-kill-line-SCREEN

IF the goal is to delete text
  AND the deletion spans the entire line
THEN use the kill-line method.
```

As an example of the latter type, the next rule checks for superfluous space following a deletion:

```
PRODUCTION too-much-space-SCREEN

IF the goal is to check for space following a deletion
  AND the cursor is positioned on a space character
  AND the character to the left is also a space character
THEN delete the previous character.
```

MT-EMACS. These rules simply specify the command bindings for the four deletion operators in *EMACS*. We have seen several examples of these kinds of rules already.

MT-PERVERSE. These four rules are the counterparts of those in MT-*EMACS*.

This completes our description of the rule sets used to simulate behavior in the editors. One remaining task is to identify the rules that in our view are practiced by the typing control group from Experiment 1 and are transferred to *EMACS*. Typing control subjects must acquire edits from the manuscript and, therefore, practice two high-frequency rules from LL-GENERAL. As for MT, subjects practice only a single rule from the SCREEN category, which states that if the goal is to insert text, then set as a subgoal to type that text (the typing interface, like the screen editors, was always in the "insert" mode).

Given this task analysis, we are now in position to make quantitative predictions of LL and MT planning time transfer for all conditions from both experiments. The method is simply to sum the production frequencies for a particular editor and then figure the percentage of firings that involved known rules. Of course, the known rules are defined by our various categories and differ depending on the particular transfer condition being modeled. For example, to calculate percentage transfer for LL from *ED* to EDT, one would use the following formula:

$$\%_{\text{tran}} = \frac{f_{\text{LL-GENERAL}} + f_{\text{LL-LINE}}}{f_{\text{LL-GENERAL}} + f_{\text{LL-LINE}} + f_{\text{LL-EDT}}} \times 100 \qquad (3)$$

where f_x is the sum of production frequencies for category x. In this case, the numerator represents the rules shared by the line editor LL components, and the denominator represents the entire set of rules required for LL in EDT.

Instantiating this formula, we get:

$$\frac{96 + 55.4}{96 + 55.4 + 73.4} \times 100 = 67\%$$

Whereas we use Equation 3 to generate our theoretical predictions, it is important to note that we used a different equation (Equation 1) to measure transfer empirically. At first blush, the equivalence of Equations 3 and 1 is not apparent. However, we have shown (see Singley & Anderson, 1988, for a detailed derivation) that Equation 1 can be reduced to Equation 3 under the following set of assumptions:

1. All productions take roughly the same amount of time to learn and execute.
2. Exposure to the training task does not affect the structure of the transfer task.
3. Common elements occur with roughly the same frequencies in the training and transfer tasks.
4. Measures of subject performance are aggregated over roughly the same number of trials in training and transfer tasks.

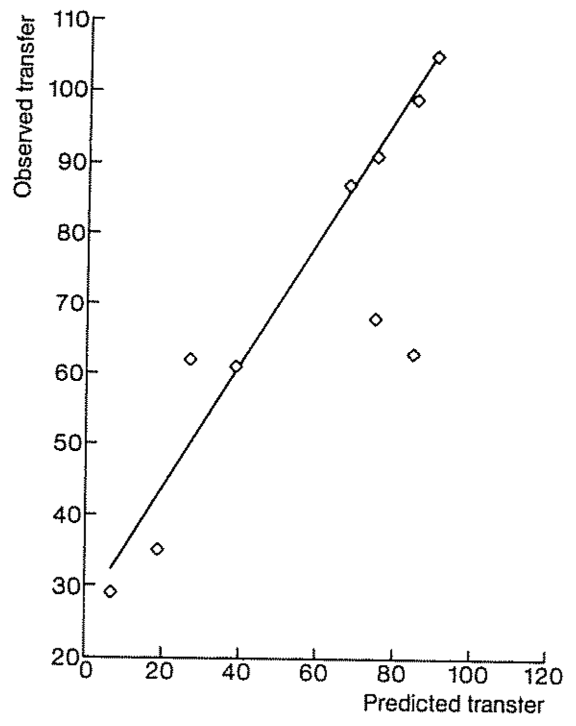
Needless to say, violations of these assumptions may account for certain discrepancies between predicted and observed values.

Figure 17 presents predicted and observed transfer percentages for the various conditions from both experiments in tabular form, and Figure 18

Figure 17. **Observed and predicted transfer percentages for LL and MT planning time in the two experiments.**

Component	Training Editor	Transfer Editor	% Transfer Predicted	% Transfer Observed
LL	*ED*	*EDT*	68	87
	EDT	*ED*	75	91
	LINE	*EMACS*	39	61
	TYPING	*EMACS*	19	35
	EMACS	*PERVERSE EMACS*	75	68
MT	*ED*	*EDT*	90	105
	EDT	*ED*	85	99
	LINE	*EMACS*	27	62
	TYPING	*EMACS*	7	29
	EMACS	*PERVERSE EMACS*	85	63

Figure 18. **Graphical depiction of the data in Figure 17.**



presents that same data in graphical form. Generally, we see a good fit between predicted and observed values; the correlation between these sets of points is .85. However, upon closer inspection, we see that in 8 out of 10 cases, we are underpredicting the amount of observed transfer, and in the remaining two, we are overpredicting. Interestingly, the eight cases of underprediction are all drawn from Experiment 1, and the two cases of overprediction concern LL and MT transfer from *EMACS* to *perverse EMACS*.

If one excludes the two points involving transfer from *EMACS* to *perverse EMACS*, the correlation becomes .98. This represents an almost perfect linear relationship between our production-overlap predictions and empirical measures of transfer. This linear relationship is as follows:

$$\mathrm{O} = .26 + .88\mathrm{T} \tag{4}$$

where O is the observed transfer and T is the theoretical prediction. Unfortunately, such an equation makes little sense because it predicts greater than 100% transfer when T = 1, that is, when there is total production overlap.

One simplifying assumption made throughout this analysis is that all productions have equal cost in terms of learning, and therefore, contribute equally to transfer. However, given the systematic bias in our predictions, it may be that we are assigning inappropriate weights to the components of transfer. Specifically, it appears that we are underestimating the contribution of the general components and overestimating the contribution of the specific components. Looking at the low extreme case, we find ourselves predicting about 13% transfer in the case of typing to *EMACS* and observing about 32%. The typing condition and *EMACS* share a high-frequency production that encodes edits from the manuscript. It is perhaps reasonable that there is more to a production that interprets the edit marks on a line than many of the productions specific to *EMACS*, for example, a production that issues the ↑f keystroke. Looking at the high-extreme case, we predict 79% transfer between the line editors and observe 96%. This suggests that the productions specific to the line editors are being overweighted. Again, it is not unreasonable to suppose a production that issues the p command is less costly to learn than a production that decides among methods.

To identify the appropriate weightings, we could rewrite our linear transfer equation as:

$$\mathrm{O} = \frac{\mathrm{T} + .30}{1 + .30 - .16} = .26 + .88\mathrm{T} \tag{5}$$

where the + .30 represents the extra emphasis needed for the general productions and − .16 represents the lesser emphasis needed for the terminal productions unique to an editor. Without these adjustments, Equation 5 reduces to the simple equation O = T. In summary, we are encouraged by the close linear relationship between observed and predicted transfer that exists when we exclude *perverse EMACS* from our analysis. The fact that we do not get the simple equation O = T probably reflects the fact that different productions deserve different weightings in the transfer equation.

The well-behaved character of the non*perverse EMACS* points in Figure 18 makes the overprediction of the *perverse EMACS* points all the more striking. One is tempted to conclude that here is our first evidence of true procedural interference between the screen editors in the planning components. Note that this is negative transfer only in the relative sense, however. The overwhelming effect is positive, just not as positive as we had predicted.

5. GENERAL DISCUSSION

The results of our two experiments largely support an identical elements model of transfer based on a production system representation of cognitive skill. In the first experiment, the relative magnitudes of transfer observed were consistent with detailed measures of production system overlap between editors. In addition, localized transfer sites were hypothesized and identified through a series of empirical microanalyses. In the second experiment, specific transfer predictions based on the differential practice of components were tested and confirmed. Using a classic interference paradigm, little evidence was found for negative transfer in the usual sense.

Additional Transfer Components. Although our identical elements models were quite successful at making relative predictions, they were less able to predict the magnitude of transfer in absolute terms. Most notable was the tendency to underpredict the magnitude of transfer. We have already proposed an explanation for this effect, although other explanations exist. We suggest one additional source of transfer that represents a significant complication to our simple production system analysis. This is the role of declarative knowledge in procedural transfer. It is our view that declarative knowledge of a special sort contributed to transfer between the editors, although we gathered no data in our experiments to support this claim. Our lack of data on this point is primarily due to our choice of methodology: Keystroke protocols are not very informative about the role of declarative knowledge in skill acquisition and transfer. However, other researchers (Bott, 1979; Mack et al., 1983) have taken verbal protocols of subjects learning to use an editor and have found that critical to the initial performance of the skill is a rudimentary mental model of the editor as an interactive device. For example, a novice may be ignorant of such basic facts as:

1. Text editing is an interactive dialogue between user and computer.
2. The user issues commands to the computer, not vice versa.

3. The computer allows for mistakes, and the user must detect and correct them.

Facts such as these play a useful role early on by supplying a kind of high-level search control which facilitates a subject's structuring of the task. However, although such knowledge is certainly integral to text editing, subjects quickly get beyond this stage and move on to the real order of business: learning the particular procedures of a particular editor. Because other researchers had already done quite well in the study of a subject's initial exposure to an interactive device, we chose to take a more extended view of learning in our experiments.

General Versus Specific Transfer. Our subjects were quite resourceful and opportunistic in exploiting areas of production overlap in the transfer task. In some cases, they identified areas of commonality in the performance of the task that we had overlooked as expert observers, as mentioned previously. No doubt this was partly due to our multiple-trials methodology; subjects were given every opportunity to discover similarities.

Despite the seeming intelligence of our subjects, our results suggest that transfer was rather local and task-specific. The question naturally arises as to whether our theory has any place for the transfer of some task-independent component such as the various weak methods documented by Newell and Simon (1972). The weak methods are invariably cast in procedural terms and would, therefore, be an easy addition to any production system model of skill. However, given our analysis, although the weak methods often play a vital role in the initial performance of a skill, they play virtually no role in transfer. The reason for this is not profound but rather a simple consequence of how transfer is measured. If a component is already well-practiced before training begins, learning will be negligible and the component will simply drop out of the transfer equation. This is true even when the component is a significant part of the transfer task, which is the case in our experiments with the subskill of typing. Because our subjects were already skilled typists before the experiment began, learning was negligible, and, as a result, the typing component had no impact on transfer. Alternatively, if our subjects had known nothing about typing, we would have observed massive learning and subsequent transfer of this subskill. This point highlights the fact that all skills are learned not in isolation but rather against a backdrop of well-practiced support skills which go virtually undetected in any measurement of learning and transfer. In the typical psychology experiment using adults, the weak methods fall into this latter class of support skills. Presumably, once acquired and automated, nothing significant is gained through their repeated practice other than a slight speedup. This may partly explain why the transfer of general problem-solving skills has been so difficult to detect in adults (Jeffries, 1978).

Indeterminacy in Transfer Predictions. Putting aside the difficulties resulting from inaccurate task analyses, quantitative predictions are still very difficult when studying a complex skill such as text editing in full-blown form. One reason for the difficulty is that, in an editor with close to full functionality, subjects are confronted with many choices concerning methods for accomplishing particular edits. One subject may fixate on a method in a training editor that figures prominently in a transfer editor, whereas another may fixate on a method that has no role in the transfer editor at all. As a result, the first subject exhibits more transfer than the second. This difficulty arises whenever methods and/or strategic knowledge play a prominent role in a skill. Unless subjects are forced to practice certain components, transfer will be indeterminate to some degree.

Negative Transfer Revisited. Of course, the issue of negative transfer cannot be decided by a couple of experiments. However, our results suggest that interference in the classic sense is quite hard to achieve in the realm of cognitive skills. What little negative transfer we observed was explained largely as the positive transfer of a nonoptimal method which blocked the acquisition and use of a new method. The fact that declarative interference is well-documented (Postman, 1971) but procedural interference is not suggests another possible distinction between these two types of knowledge (Anderson, 1983). Ultimately, this is additional evidence for the declarative-procedural distinction in knowledge representation.

Acknowledgments. We thank Jeff Shrager for his help in programming the experiments, and also Peter Pirolli, John Karat, and especially Peter Polson for insightful comments on earlier versions of the manuscript.

Support. This research was supported by the Personnel and Training Research Programs, Psychological Services Division, Office of Naval Research, under Contract No. N00014-84-K-0064.

REFERENCES

Anderson, J. R. (1983). *The architecture of cognition.* Cambridge, MA: Harvard University Press.

Angell, J. R. (1908). The doctrine of formal discipline in the light of the principles of general psychology. *Educational Review, 36,* 1-14.

Bott, R. A. (1979). *A study of complex learning: theory and methodologies* (Report No. 7901). La Jolla, CA: University of California at San Diego, Center for Human Information Processing.

Bower, G. H., & Hilgard, E. (1981). *Theories of learning.* Englewood Cliffs, NJ: Prentice-Hall.

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

Douglas, S. A. (1983). *Learning to text edit: Semantics in procedural skill acquisition.* Doctoral dissertation, Stanford University, Stanford, CA.

Douglas, S. A., & Moran, T. P. (1983). Learning text editor semantics by analogy. *Proceedings of the CHI '83 Conference on Human Factors in Computing Systems,* 207-216. New York: ACM.

Ellis, H. C. (1965). *The transfer of learning.* New York: Macmillan.

Gosling, J. (1981). *Unix EMACS user manual.* Pittsburgh: Carnegie-Mellon University, Computer Science Department.

Introduction to the EDT editor. (1982). Marlborough, MA: Digital Equipment Corporation.

Hayes, J. R., & Simon, H. A. (1977). Psychological differences among problem isomorphs. In N. J. Castellan, D. B. Pisoni, & G. R. Potts (Eds.), *Cognitive theory* (Vol. 2, pp. 21-41). Hillsdale, NJ: Lawrence Erlbaum Associates, Inc.

Jeffries, R. (1978). *The acquisition of expertise on missionaries-cannibals and water jug problems.* Doctoral dissertation, University of Colorado, Boulder.

Katona, G. (1940). *Organizing and memorizing.* New York: Columbia University Press.

Kernighan, B. W., & Pike, R. (1984). *The UNIX programming environment.* Englewood Cliffs, NJ: Prentice-Hall.

Kieras, D. E., & Bovair, S. (1985). *The acquisition of procedures from text: A production system analysis of transfer of training* (Report No. 16). Ann Arbor: University of Michigan.

Linn, M. C., & Fisher, C. W. (1983). The gap between promise and reality in computer education: Planning a response. In *Making our schools more effective: A conference for California educators.* San Francisco: ACCCEL.

Luchins, A. S. (1942). Mechanization in problem solving. *Psychological Monographs, 54*(248).

Mack, R., Lewis, C. H., & Carroll, J. (1983). Learning to use word processors: problems and prospects. *ACM Transactions on Office Information Systems, 3,* 254-271.

Moran, T. P. (1983). Getting into a system: External-internal task mapping analysis. *Proceedings of the CHI '83 Conference on Human Factors in Computing Systems,* 45-49. New York: ACM.

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

Orata, P. T. (1928). *The theory of identical elements.* Columbus: Ohio State University Press.

Papert, S. (1980). *Mindstorms: Children, computers, and powerful ideas.* New York: Basic Books.

Polson, P. G., & Kieras, D. E. (1985). A quantitative model of the learning and performance of text-editing knowledge. *Proceedings of the CHI '85 Conference on Human Factors in Computing Systems,* 207-212. New York: ACM.

Postman, L. (1971). Organization and interference. *Psychological Review, 78,* 290-302.

Sauers, R., & Farrell, R. (1982). *GRAPES user's manual* (Tech. Rep. No. ONR-82-3). Pittsburgh, PA: Carnegie-Mellon University.

Simon, H. A. (1968). *The sciences of the artificial.* Cambridge, MA: MIT Press.

Singley, M. K., & Anderson, J. R. (1985). The transfer of text-editing skill. *International Journal of Man-Machine Studies, 22,* 403-423.

Singley, M. K., & Anderson, J. R. (1988). *The transfer of cognitive skill.* Manuscript in preparation.

Smith, S. B. (1986). *An analysis of transfer between tower of Hanoi isomorphs.* Doctoral dissertation, Carnegie-Mellon University.

Thorndike, E. L., & Woodworth, R. S. (1901). The influence of improvement in one mental function upon the efficiency of other functions. *Psychological Review, 8,* 247-261.

HCI Editorial Record. First manuscript received October 28, 1986. Revisions received July 26, 1987, and December 3, 1987. Accepted by Peter Polson—*Editor*

APPENDIX: KEYSTROKE PARSING PROCEDURE

Figure 19 shows the parsing algorithm in pseudocode. Before parsing any commands, the program initializes the goal stack for the trial being parsed. Recall that each trial consists of six edits randomly distributed in an 18-line file. Also recall that there are 12 distinct types of edits defined by crossing the editing operations insert, replace, and delete by the data objects character, word, string, and line. Thus, an instance of a goal stack[4] is:

(1) insert line, line 1, character 1
(2) insert word, line 7, character 31
(3) delete character, line 10, character 4
(4) replace line, line 11, characters 1-44
(5) delete string, line 13, characters 40-50
(6) replace word, line 14, characters 31-41

Given a goal stack such as this, the parsing program has certain well-defined expectations about the sequence of commands executed by subjects. First, subjects enter some sequence of LL commands to position themselves for the first goal. Then subjects execute a series of MT commands to fix the mutilation. Subjects continue this LL/MT alternation until all six goals have been satisfied. In this way, the parser segments the keystroke data into six major episodes, each containing LL and MT subcomponents. The keystrokes and time intervals of each of these episodes are attributed to each of the six goals. The simplifying assumption here is that subjects invariably proceed from top to bottom in a file and do not backtrack. This assumption is fairly plausible given that subjects are positioned at the top of the file at the start of each trial. As shown in Figure 4, this assumption is in fact born out by the success of our parsing algorithm.

There are two major modules in the program, labeled *normal* and *limbo.* In most straightforward cases of parsing, control remains in the *while* loop of the

[4] This is in fact the goal stack for the trial shown in Figure 2. Goal stack information is saved for each trial by the text mutilation program that introduces the errors originally.

Figure 19. **The keystroke parsing algorithm in pseudocode. Control starts in the *normal* module.**

```
initialize goalstack

NORMAL:
  while goalstack is non-empty
    read goal on top of stack
    parse locate line sequence and update position
    if current position is close to goal position
      parse modify text sequence
      pop goal from stack

    else goto LIMBO

LIMBO:
  while there are keystrokes left to be parsed
    copy goalstack to temporary stack
    while temporary stack is non-empty
      read goal on top of temporary stack
      if current position is close to goal position
        copy temporary stack to goal stack
        attribute 'lost' sequence
        goto NORMAL

      else pop goal from temporary stack

    parse current command and update position
    update goal stack if line numbers have changed
```

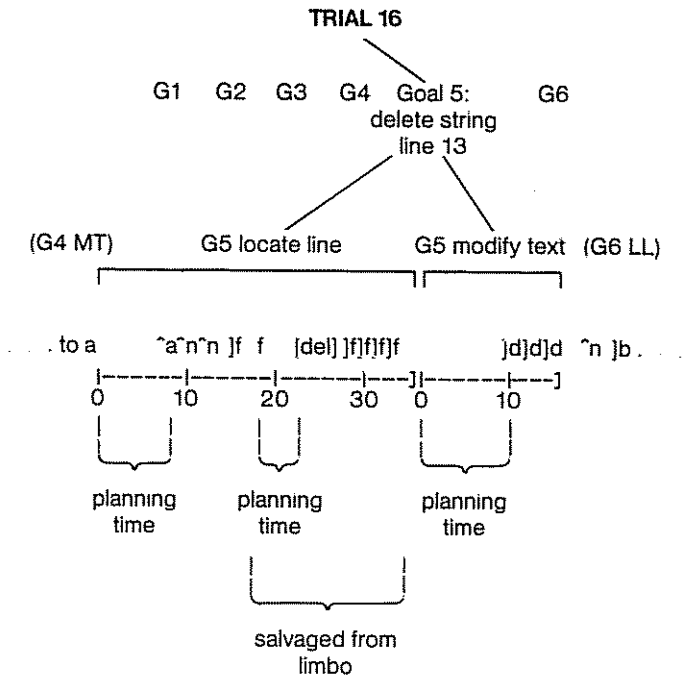
normal module, reading LL/MT sequences until all goals are satisfied and no more keystrokes remain. However, if after reading a LL sequence the simulation program is out of position[5] for the top goal on the stack, an error condition results and control is passed to the *limbo* module. The subject has deviated from the expectations of the parser and keystrokes can no longer be attributed reliably to goals.

In most cases, the parser goes into limbo because subjects have done one of three things: inadvertantly modified the wrong text, skipped a goal, or backtracked to remedy a previous mistake. Our strategy at this point is to temporarily reserve judgement on the commands being parsed until the subject is once again in position to satisfy any of the goals remaining on the stack. We then resume normal parsing after trying to salvage any or all parts of the lost sequence. We use the following heuristics to attribute the lost sequence:

- If the parser finds itself in position for the goal that was on the top of the stack when limbo was first entered, the entire lost sequence is attributed to the LL subcomponent of that goal.

[5] Out of position is defined as being on the wrong line in the line editors and more than 10 characters away from the locus of the modification in *EMACS*. The locus is a single character for insert operations, but usually a range of characters for replace and delete operations.

Figure 20. **A timeline of keystroke data showing the application of the parsing algorithm. This excerpt is based on the satisfaction of Goal 5 in Figure 1.**



- If the parser finds itself in position for a later goal, all but the last LL sequence in the lost sequence is thrown away. This last LL sequence is attributed to the LL subcomponent of the later goal. This goal is made the current goal and parsing continues normally.

Measuring Planning and Execution Times. The last step in the parsing analysis is to partition the time intervals attributed to the various LL and MT sequences into their planning and execution components. A LL or MT sequence is composed of one or more commands, each of which has its own planning and execution component. We take the planning component of a command to be the sum of two types of pauses: the single long pause that extends from the last keystroke of the previous command to the first keystroke of the current command and the various shorter pauses that may or may not separate keystroke bursts within the command. (Times between keystrokes are classified as pauses if they are longer than 2 sec.) Given this definition, the execution component is merely the time from the first to last keystroke of the command minus any interkeystroke pauses.

Figure 20 shows a timeline of keystroke data and its breakdown into components. This keystroke excerpt is taken from a hypothetical subject using *EMACS* to perform the fifth modification (delete string, line 13) from the trial shown in Figure 2. To put this excerpt into context, included are the last few keystrokes from the MT component of the previous goal, replace line 11. The clock for the current goal starts when the last key is struck from the previous goal. The LL component starts with a long pause followed by ↑a, which positions the cursor at the beginning of line 11, and then ↑n↑n, which moves the cursor down two lines. Now the subject must move across line 13 to the position of the string to be deleted. The subject begins with]f, which moves the cursor ahead one word. This is followed by a standard, alphanumeric f, which modifies the text and breaks the sequence of locate line commands. (In *EMACS*, all alphanumeric characters are inserted at the current cursor position.) As intelligent observers, we surmise that the subject actually intended to execute another]f command and merely forgot to strike the escape key, a common novice error due to a misgeneralization of the control and shift keys.[6] At this point in the parsing, attribution of keystrokes is temporarily suspended, because a modification has been made away from the current goal site. As the parser continues in limbo, the subject pauses, deletes the mistaken f, and ends the LL sequence with four]f commands. The parser now awakens, because the subject is in position to satisfy the current goal. Using the first heuristic just outlined, the parser attributes all of the lost keystrokes to the LL component of the current goal. The MT component begins with a short pause followed by three]d commands, which delete the three words in the string. This concludes the parsing for this goal, as the subject moves on to the next goal site.

As should be apparent from the preceding description, there are two shortcomings with our parsing analysis. The first is that any time spent looking at the screen to verify that a modification has been done correctly is misattributed to the planning component of the following LL sequence. The second is that any time spent encoding the edit from the manuscript is also buried in the planning component of the LL sequence. Complicating this second point is our informal observation that subjects sometimes looked at the manuscript several times in the course of an edit. Although we could have better separated the encode edit component by videotaping, this route was impractical given the quantity of our data.

[6] The escape key differs from the control and shift keys in that it must be released and depressed between consecutive commands