Scroll Editing: An On-Line Algorithm for Manipulating Long Character Strings

MARY ALLEN WILKES

Abstract—An algorithm that runs on a 2048-word LINC provides efficient on-line editing of character strings virtually unlimited in length. Fixed-address LINC tape holds the character sequence in the manner of a scroll. Edited characters are spliced directly in or out of the scroll as it moves across a display scope under the viewer's control. A 512-character "playground" created at the splice point provides sufficient ease to permit changing the scroll contents dynamically, and thereby simplifies several problems commonly associated with on-line editing. Compensatory inserting and deleting are practical. Inserted characters require no special identification and scroll maintenance is automatic. Editing commands for simple editorial functions and editorial text identifiers are eliminated, and the number of characters that can be inserted anywhere is limited only by the total length of the scroll. Line numbers, if provided, are resequenced automatically as the scroll contents change. As little as 2 percent of the scroll is manipulated in the memory at a time. Despite the relatively slow transfer characteristics of the tape, performance is satisfactory on a LINC for scrolls up to 23 040 characters and is not strongly dependent on the size of the playground.

Index Terms—LAP6, LINC, on-line editing, scope editing, scroll editing, tape editing.

Introduction

SCHWARTZ has remarked that "the process of editing code on-line is considered by some to be the heart of a good on-line system" [1]. Selecting an algorithm to govern an on-line editor involves recognizing that editing consists first of identifying an editorial point of interest in a collection of characters. It then entails only two things: adding characters and/or removing characters at that unique point. An algorithm that provides a simple, direct, and efficient way to handle these elementary activities can be used to construct any editorial function whatever. This fact, although frequently acknowledged [2], [3], has been enthusiastically obscured in many computer environments.

Although there are several functional descriptions of on-line editors [2]–[11], their data-handling algorithms tend to be infrequently documented. The focus of any algorithm is its character-insertion mechanism. Editors that manipulate characters in literal strings most frequently use a "deferred storage" technique, which saves information being inserted in a temporary buffer, and defers storing it in the text itself until some condition has been satisfied. The disadvantage of handling the data twice has other implications that can appear, for example, in the form of special editing commands required of the user or as a limitation on the size of the buffer. Extending the buffer to a peripheral device removes this limitation, but the task of eventually inserting the material into the text may be aggravated.

Semi-list structures, popularly used with disks [2], [7]–[11], avoid the insertion problem at the expense of text fragmentation and increasing overhead, without eliminating the problem of overflow onto the peripheral device. Except for a few applications [2], [7], most find no inherent advantage in the segmented text and generally resort to resequencing or even translating the list structure into a literal string.

The scroll-editing algorithm uses what might be called a "direct storage" technique in which the data are handled once as they move from their input source into the text string, which itself is directly continuous between the core memory and a randomly addressable peripheral device. Motivated by experience with the inefficiencies of the deferred storage technique [12], the algorithm described here has, since its initial use in 1966 [13], been found far superior.

A machine such as the LINC [14] is excellently suited to text manipulation, despite its 2048-word memory. Display scope and keyboard are standard items, as is a two-unit magnetic tape transport whose pocket-sized premarked tapes are randomly addressable by block number in a manner similar to the TX-2 [15] or the ATLAS computer [16]. The LINC searches for a requested block by moving the tape in either direction.

Text, or manuscript, is held on a LINC tape that is treated as a scroll wound onto the two tape hubs. Motion of the scroll, to expose different parts of the manuscript on the scope, can be directed from the keyboard. As a reading technique, this method of text presentation is a logical one for scope editors.

The concept of scroll *editing*, however, means that the scroll itself, theoretically unlimited in length, can be edited anywhere and to any extent by writing directly on the scroll or by erasing information directly from the scroll at the unique points of interest. The scroll remains homogeneous, although its contents, and probably its length, change dynamically with every editorial correction.

Functional Implications

Scroll editing is an integral part of the on-line LINC system LAP6 [17]. Of the editors cited, LAP6 is functionally closest to Tolliver's TVEDIT, and was similarly influenced by a desire to let the user's interaction with the displayed text be governed by the explicit context-dependent nature of his

Manuscript received August 26, 1969; revised March 2, 1970. This work was supported by the Division of Research Facilities and Resources of the National Institutes of Health under Grant FR-218 through Washington University, St. Louis, Mo.

The author is with the Computer Systems Laboratory, Washington University, St. Louis, Mo.

environment. It is an environment in which highly structured commands can be, and more often should be, eliminated in favor of the user's natural response to the visually obvious. The philosophy is ably expounded by McCarthy *et al.* [11].

It is not the purpose here to argue for a specific functional description. The essentials of the algorithm are the same regardless of what features are built out of them. The editing functions that have been tried include single character, line, and multiple-line manipulations for insertions, deletions, or combinations, and are all straightforward both to implement and to use. LAP6 commands that operate on manuscripts are streamlined. A single ADD MANUSCRIPT command, for example, is used to move a manuscript onto an empty scroll, append a filed manuscript to the scroll manuscript, or insert one manuscript in the middle of another. Its counterpart, SAVE MANUSCRIPT, is used for the complementary functions. Manuscripts can be broken into any pieces for rearranging, filing, printing, etc. [18].

Scroll Motion

A LAP6 manuscript is any collection of character codes generated from any source. A "current manuscript" is held in consecutive blocks of the *scroll area* on the tape. A block is simply the transfer unit between tape and memory; the characters carry smoothly across block boundaries, producing a continuous string.

The scroll is moved, either by the user or automatically under program control, between editorial points of interest, i.e., from one "requested scroll position" to another. A requested scroll position can be any conveniently definable position between two characters. In Fig. 1 the scroll position is between the last character of line 340 and the first character of the undisplayed line 341. One version permits positioning by matched character string, line number, or in directional increments of single characters, lines, or frames. The mechanism itself is less important than the ability to move the scroll easily and directly to other positions. In most cases forward and backward pushbuttons would be more nearly ideal.

Specific Implications

Some features characterize all editors. The way LAP6 handles the following fundamental situations is either directly implied by the editing algorithm or greatly simplified by it.

1) At any requested scroll position, keyboard input can be typed directly to the scroll. No INSERT function or function bounds are needed; the user does not "tell" the machine that he is editing. Starting with Fig. 1, for example, added information automatically becomes line 341, then 342, etc.
2) At any scroll position any amount of material can be edited in or out without interruption requiring user action.

Fig. 1. The display.

3) The scroll can be moved in either direction and edited in any order.
4) Moving the text and identifying the editorial point of interest can be synonymous.
5) The text can have any format; if it is line-oriented, lines can be of any length.
6) Line or sequence numbers, if provided, are automatically renumbered. The line numbers are not part of the manuscript; in Fig. 1 if the user adds two lines and then moves the scroll forward, the former line 341 will have the number 343.
7) The total storage required is just that to hold the characters; specifically, copies of the text are not required.
8) The core memory required is minimal; the LAP6 scroll uses a maximum of 768 words.
9) The simple manuscript structure permits manuscripts to be used, generated, and edited easily by programs other than the system editor.
10) Scroll "maintenance" is automatic; the user is not responsible for, e.g., "packing," or moving the scroll to special positions (typically the end or the beginning).

The Algorithm

The editing algorithm was first proposed by Stucki and Ornstein in discussions with the author during March 1965. Despite subsequent refinement, the playground concept and the organization of LINC tape and memory to accommodate it are largely their creation.

The algorithm is appropriate for any randomly addressable storage device such as a disk, a premarked magnetic tape, or, if necessary, core memory. Very little core memory is required, however, and the size of the LINC memory, 2048 12-bit words, greatly motivated articulation of the algorithm. The total manuscript size and manuscript line length are, however, virtually unlimited.

The Playground

The Stucki–Ornstein algorithm is based on the concept of a *playground*, created by separating the character string in the memory at the requested scroll position to provide a free space for additional characters coming from the keyboard. Characters, picked up or discarded in the playground, are "spliced" in or out of the scroll as it moves

through the memory from one tape reel to the other. The playground, once created, continues to "track" the changing scroll position as the scroll moves.

Creating the Playground

Fig. 2 shows a manuscript on the tape in blocks 1–7 of the scroll area. BS, WS, and CS are 256-word memory sections corresponding in size to scroll blocks. If block 4 contains a requested position, say line 340, block 4 is read into the memory and the string broken between lines 340 and 341. In Fig. 2, X is at the end of line 340, and Y at the beginning of line 341. Block 3, read into the buffer section BS, provides continuity for the display, which is as in Fig. 1.

The original string is now treated as two separate strings. The first, or *working string*, is contained in blocks 1, 2, 3, and the working section, WS, up to point X. The second, or *continuation string*, begins at point Y in the continuation section CS, followed by blocks 5, 6, and 7.

The shaded area in Fig. 2 represents the playground, an expansion into the memory of the scroll area on the tape. The playground happens to be the same size as a memory section or scroll block, but this is not required. It could be larger or smaller, its purpose being only to create "some" extra space at the editorial point of interest.

Positioning the Scroll

If starting with Fig. 2 a new scroll position is requested, say, line 330, characters are transferred from the working section into the continuation section (Fig. 3), until the points X and Y are between lines 330 and 331. The playground has simply shifted left. Locating forward is identical, except that the playground is shifted to the right.

Adding and Deleting Characters

New characters being added to the scroll are stored in the playground starting at point X as they arrive. X and the display move into the playground the appropriate amount keeping track of the end of the working string. Adding information shrinks the playground (Fig. 4). Other than that, there is nothing special about the new characters; the working string is still continuous from block 1 to point X.

Likewise, if information is deleted, X moves the appropriate amount to the left, shortening the working string. Deleting increases the size of the playground.

One obvious advantage of the scheme is that adding and deleting are compensatory. When initially created, the LAP 6 playground can accommodate 512 characters.[1] The total number of characters that can be added before a new playground must be created is therefore 512 *plus* the number deleted.

In either case, since the changes are made exactly where they belong in the string, no further manipulation is required by either the user or LAP 6 in order to incorporate

[1] Two character codes are stored per word.

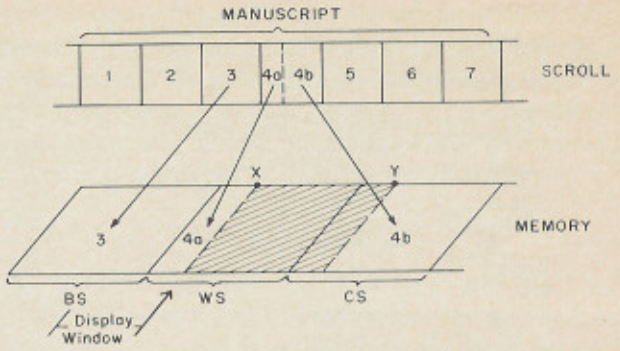

Fig. 2. Creating the playground.

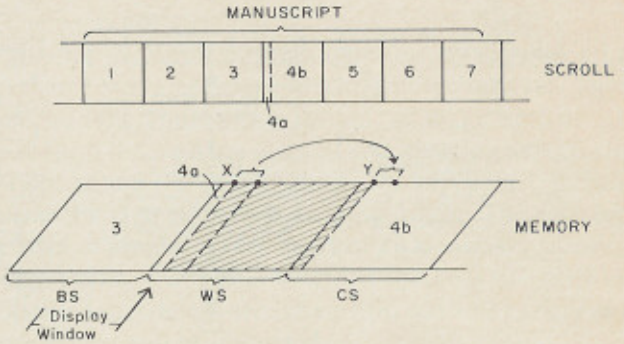

Fig. 3. Moving backward.

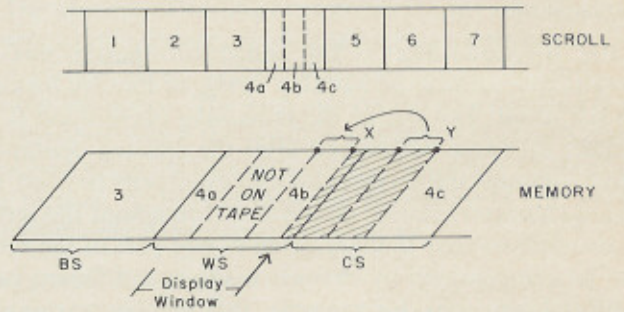

Fig. 4. Adding new information, moving forward, and splicing.

them. The relationship of the working string to the continuation string remains well defined, and there is no need to distinguish an "edited" from an "unedited" manuscript.

The algorithm as presented in Fig. 5 shows all manipulations at the simplest single character level. Without considering the memory/scroll interactions they can be summarized as follows.

1) Moving the scroll in either direction shifts the playground left or right, moving X and Y in parallel.
2) Adding or deleting information causes the playground to shrink or expand, since only X is moved. Point Y and the continuation string are temporarily ignored.

In what follows, combinations of these two activities are considered in connection with line numbers, section boundary conditions, and the full playground situation. Further illustrations are given in [13] and [19].

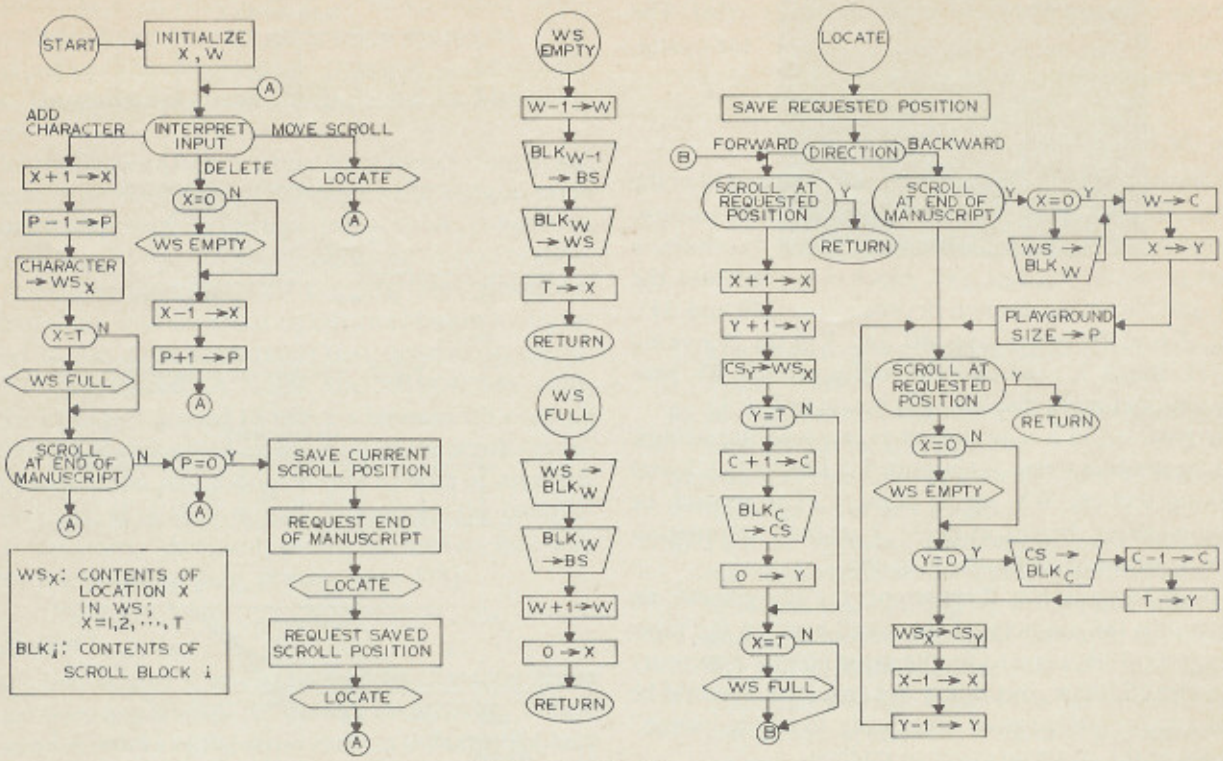

Fig. 5. Simplified flow diagram.

Splicing

Fig. 4 follows Fig. 2. The added information NOT ON TAPE is spliced into the working string by moving forward on the scroll. The size of the remaining playground is, of course, the same before and after the splicing operation. Thus, although the playground may eventually be filled, the added characters can be distributed throughout the scroll. It also follows that changes made to one part of the scroll can be compensated by changes made to a different part. Editorial functions, for example, that involve rearranging large sections of the text are readily accommodated.

Numbering the Lines

Since the algorithm does not require that the lines have fixed identifiers, line numbers can be provided and renumbered automatically by using a single counter to monitor a current line number. The counter is decremented whenever a line terminator is removed from the working string, whether from deleting or locating, and incremented whenever a line terminator is added to the working string. Thus, as character input is interspersed with forward locating, "old" lines are automatically renumbered as they are brought from the continuation string onto the scope.

Section Boundaries

The memory/scroll interaction required when either WS or CS is full or empty can be determined from Fig. 5. W and C on the chart represent scroll block numbers associated with the working and continuation strings, respectively. W is always the number of the scroll block in which WS will next be written if it fills up. C, initially set equal to W, is established every time a playground is created.

WS can be filled ($X=T$) or emptied ($X=0$) by combinations of editing and locating. CS, however, is affected only by locating. The scroll block pointers W and C therefore change independently. For the situations of Figs. 2–4, W and C both equal 4. Thus if WS fills up following Fig. 4, it will be written in block 4 (Fig. 6). If it empties, block 3 will be read into WS and block 2 into BS. The procedure when CS fills up ($Y=0$) or empties ($Y=T$) is similar.

When a transfer is made in or out of either *WS* or *CS*, the string pointer X or Y is reset to the opposite boundary of the section to keep the string continuous between tape and memory. In Fig. 6 the working string now continues from block 4 to X in WS.

Since the playground always begins at X, the shaded area in Fig. 6 represents only the size of the remaining playground. (Permitting X to continue into CS requires for some manipulations that CS and WS switch roles, which is programmatically awkward.) Also, since the memory contains the same *amount* of information NOT ON TAPE in Fig. 6 that was entered as "new" information in Fig. 4, the size of the playground is still the same.

The information not on the scroll is therefore not necessarily inserted information. In Fig. 6, for example, it is the remainder of the 4c segment of the original contents of block 4.

When the transfer is made using a LINC tape, the display is interrupted for about 0.1 second. Since block 4 was the last block read (Fig. 2), the tape is properly positioned and there is little travel time loss.

Deleting can cause the two strings to become widely separated. Fig. 7 follows Fig. 2 by first deleting until the working string is continuous only between block 1 and WS,

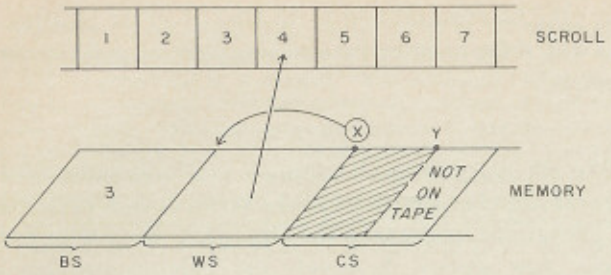

Fig. 6. Working section full.

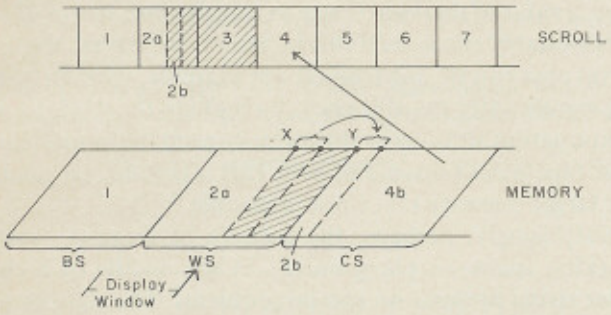

Fig. 7. Strings separated by deleting and playground shifted.

and then locating backward by an amount 2b. Since the deleted characters are simply characters that need not be rewritten, the additional playground created is represented by the shaded area on the scroll in Fig. 7. The size of the playground is, of course, the size of this shaded area plus 512 (from Fig. 2).

The backward locating operation in Fig. 7 fills CS. As it is written in block 4, segment 2b is moved from block 2 to block 4, and the additional playground is shifted left into blocks 2 and 3. The shift is identical to the playground shift in Fig. 3, but it is done on the scroll.

Joining the Strings

Moving the scroll forward to the end of the manuscript is all that is required to join the two strings. When the last character is moved from Y to X, the working string is the entire string and the playground is "pushed off" the end.

The LAP6 user is never required to join the strings. He states system commands at any time, regardless of the position of the scroll or the extent of editing. If the strings need to be joined (e.g., to file the current manuscript), LAP6 joins them automatically. This avoids unnecessary tape handling since under program control the decision to join the strings can be based only on the extent of editing, and not influenced by the scroll position.

Playground Full

The playground is full when 512 characters in the memory are NOT ON TAPE, and W equals C. At this point, information about the current scroll position is saved and the two strings are joined automatically as described above. A new playground is then created at the former scroll position. Since the result on the scope and in the memory is the same before and after this operation, there is no change from the viewer's standpoint.

It must be emphasized that a full playground in no way represents an awkward situation or interferes with the system operation. The procedure for handling it is identical to executing two locate requests, except that it is automatic. The programming overhead incurred is therefore minor; in the LAP6 case it is about 30 machine instructions.

Efficiency

The standard configuration of LAP6 has a 45-block scroll that accommodates 23 040 characters. The size has been found more than adequate for most LINC applications, although a few have used scrolls as long as 270 blocks. Although the algorithm is unrelated to scroll size, its efficiency, of course, is not.

Moving the scroll through the memory requires a little less than 1 second per tape block. This includes reading the block into the continuation section, writing it back onto the scroll from the working section, and checking the transfers. The WRITE and CHECK operations, for which the LINC has to reverse the tape twice, are the most costly. The LINC tapes are read at regular travel speed of 23 blocks per second.

In practice, the user is aware of frequent but brief tape motions. Long delays are infrequent, due to a variety of influences that will be discussed.

Editing Efficiency

The absence of editing commands for the simple editing functions of inserting and deleting reduces both typing time and errors. These explicit activities probably cannot be further simplified. The response on the scope is always instantaneous. Since the scroll is properly positioned this is true even when the user edits across a boundary of the working section. A delayed response is possible in the editing situation only if the playground fills up.

Playground Efficiency

Although the user is unaware of the playground per se, he may notice some extra tape motion if it fills up, depending on how far the current scroll position is from the end of the manuscript.[2] In most situations, however, the playground never fills up, despite extensive editing, because of the compensatory deleting effect.

For this reason the size of the playground when initially created is not critical. Since delays due to a full playground are seldom incurred, increasing the size of the playground does not significantly affect the general efficiency.

Reducing the size of the playground, however, will eventually degrade performance. Although it is difficult to determine a smallest tolerable size, the playground should clearly not be smaller than a typical manuscript entry, i.e., than the least number of characters that might be inserted before any are deleted. Preferably it should be a multiple of that number. In LINC program preparation, for example, a manuscript entry is an instruction line of typically ten

[2] The LAP6 scroll is open ended only in one direction. Efficiency can be somewhat improved by smoothing the scroll in either direction, whichever distance is shorter. On the one-tape LAP6 system, overall tape allocation is more efficient if one end of the scroll is fixed.

characters. Our experience with the 512-character playground indicates that this multiple of about 50 times the size of an entry is probably more generous than necessary. When the data are mailing lists or bibliographies for which a typical entry is about 100 characters, performance, although poorer, is still generally acceptable to most users. Acceptability, however, is undoubtedly influenced by the fact that a full playground still requires no user action, except waiting, in order to continue editing. For most situations the author would not recommend using a playground smaller than five times the size of a typical entry.

The worst-case situation regarding the playground occurs when a number of characters is added, and none deleted, near the beginning of a long manuscript. Since the playground simply acts here as a fixed-size input text buffer, efficiency is comparable to that of the deferred storage techniques mentioned earlier. When the ADD MANUSCRIPT command is used, for example, to insert one manuscript in the middle of another, the playground repeatedly fills up. Provision is made for the LAP6 user to break the scroll into separate manuscripts, i.e., temporarily eliminate the continuation string, first.

Locating Efficiency

To reduce the delay involved in repositioning the scroll, LAP6 augments the algorithm of Fig. 5 to provide two positioning techniques. The first moves the characters through the memory at about 1 second per block as described. The second uses a "scroll index" to determine the number of the scroll block containing the requested scroll position. It then moves the tape, at 23 blocks per second, directly to the required block. A new playground is created at the destination. Although LAP6 uses a single index keyed to line numbers, indices can be keyed to any definable scroll positions such as the manuscript's internal alphanumeric order, its "tags," or section labels.

The second technique is used whenever the first technique will not change the string sequence on the tape. This is the case when there is no continuation string, or when no editorial changes have been made since the working and continuation strings were last joined. The 23 to 1 saving is considerable when the viewer is simply locating and reading, as, for example, when first finding his place in the manuscript.

When a relatively slow tape is used to hold the scroll, it is advantageous to create the playground, as is done here, without disturbing the scroll sequence. If the playground is physically embedded in the string, the two strings thus separated always have to be rejoined. The user is penalized for "browsing," and the automatic rejoining of the strings described earlier must be based on scroll position alone.

The combination of techniques keeps the locating delays from dominating the tape editing situation. If the user is only reading, the faster technique is used automatically. If he is editing, changing one character in the first block of a 30-block manuscript will cost him about 30 seconds when LAP6 rejoins the two strings. On the other hand, if extensive changes are made in the same manuscript, a process that may take him several hours, the sum of all the locating delays will, if the locating pattern is sequential, still be only 30 seconds.

Conclusions

The efficiency of scroll editing is primarily influenced by the transfer characteristics of the scroll device, rather than by limitations inherent in the algorithm. Even with a relatively slow device, however, the balance between the amount of data that can be efficiently handled and the amount of core memory required is excellent. Using a LINC tape, a ratio of total scroll size to transfer unit size of 45 to 1 is reasonably efficient. If this balance is preserved, the total scroll area can be quadrupled, for example, without affecting significantly the efficiency, by making the working and continuation sections in the memory each the equivalent of four tape blocks instead of one. This is possible on some of the larger memory LINCs now available.[3]

Surprisingly, perhaps, the concept of creating "some" working space—a playground—in the middle of a character string presents no special problems. The algorithm as programmed uses about 700 LINC instructions [20], including the display, and the investment required specifically to handle the playground itself is trivial. Nor is the efficiency especially sensitive to the playground's size. In practice the algorithm has, if anything, performed better than anticipated.

Scroll editing has been in routine use by about 2000 people on all varieties of LINCs since the summer of 1967. The technique has been found excellent for extensive editing of long character strings, and the resulting manuscript structure is simple and straightforward. As a result, it is used for data preparation in a variety of applications including the preparation of chemical formulas for molecular graphics work, bibliographies for information retrieval systems, flow charts, and a variety of programming languages.

The popularity of scroll editing is as much attributable to its operational ease as to its efficiency. The simplification of editing operations and the elimination of scroll maintenance have minimized the user's responsibility, and the editing process itself is direct, reliable, and unencumbered.

References

[1] J. I. Schwartz, "Outline programming," *Comm. ACM*, vol. 9, pp. 199–202, March 1966.

[2] S. Carmody, W. Gross, T. H. Nelson, D. Rice, and A. van Dam, "A hypertext editing system for the /360," in *Pertinent Concepts in Computer Graphics*, M. Faiman and J. Nievergelt, Eds. Urbana, Ill.: University of Illinois Press, 1969, pp. 291–330.

[3] L. P. Deutsch and B. W. Lampson, "An online editor," *Comm. ACM*, vol. 10, pp. 793–799, December 1967.

[4] M. V. Mathews and J. E. Miller, "Computer editing, typesetting, and image generation," *1965 Fall Joint Computer Conf., AFIPS Proc.*, vol. 27, pt. 1. Washington, D. C.: Spartan, 1965, pp. 389–398.

[5] H. S. Corbin and W. L. Frank, "Display oriented computer usage system," *Proc. 21st Nat. Conf. ACM*. Washington, D. C.: Thompson, 1966, pp. 515–526.

[6] G. E. Roudabush, C. R. T. Bacon, R. B. Briggs, J. A. Fierst, D. W. Isner, and H. A. Noguni, "The left hand of scholarship: Computer

[3] The micro-LINC 300, LINC-8, and PDP-12 all have memories expandable to 32 000 words, but use the same tape.

experiments with recorded text as a communication media," *1964 Fall Joint Computer Conf., AFIPS Proc.*, vol. 27, pt. 1. Washington, D. C.: Spartan, 1965, pp. 399–411.
[7] D. C. Engelbart and W. K. English, "A research center for augmenting human intellect," *1968 Fall Joint Computer Conf., AFIPS Proc.*, vol. 33, pt. 1. Washington, D. C.: Thompson, 1968, pp. 395–410.
[8] T. H. Kehl and C. Moss, "Systems programming on-line," *Computers and Biomed. Res.*, vol. 1, pp. 550–555, June 1968.
[9] H. Bratman, H. G. Martin, and E. C. Perstein, "Program composition and editing with an on-line display," *1968 Fall Joint Computer Conf., AFIPS Proc.*, vol. 33, pt. 2. Washington, D. C.: Thompson, 1968, pp. 1349–1360.
[10] B. Tolliver, "TVEDIT," Stanford University, Palo Alto, Calif., Stanford Time-Sharing Memo. 32, March 1965.
[11] J. McCarthy, D. Brian, G. Feldman, and J. Allen, "THOR—A display based time sharing system," *1967 Spring Joint Computer Conf., AFIPS Proc.*, vol. 30. Washington, D. C.: Thompson, 1967, pp. 623–633.
[12] M. A. Wilkes, *LAP3 Users' Manual*, Massachusetts Institute of Technology, Cambridge, Mass., Center Development Office Rept., August 1963.
[13] ——, "LAP5: LINC assembly program," *Proc. DECUS Spring Symp.* Maynard, Mass.: Digital Equipment Corp., 1966, pp. 43–50.
[14] W. A. Clark and C. E. Molnar, "A description of the LINC," in *Computers in Biomedical Research*, vol. 2, R. W. Stacy and B. Waxman, Eds. New York: Academic Press, 1965, pp. 35–66.
[15] R. L. Best and T. C. Stockebrand, "A computer-integrated rapid-access magnetic tape system with fixed address," *1958 Proc. Western Joint Computer Conf.* New York: American Institute of Electrical Engineers, 1959, pp. 42–46.
[16] T. Kilburn, R. B. Payne, and D. J. Howarth, "The Atlas supervisor," in *Computers: A Key to Total Systems Control, 1961 Eastern Joint Computer Conf., AFIPS Proc.*, vol. 20. New York: Macmillan, 1961, pp. 279–294.
[17] M. A. Wilkes, "Conversational access to a 2048-word machine," *Comm. ACM*, vol. 13, pp. 407–414, July 1970.
[18] ——, *LAP6 Handbook*, Computer Research Laboratory, Washington University, St. Louis, Mo., Tech. Rept. 2, May 1967.
[19] ——, *LAP6 Use of the Stucki–Ornstein Text Editing Algorithm*, Computer Systems Laboratory, Washington University, St. Louis, Mo., Tech. Rept. 18, February 1970.
[20] ——, *LAP6 Manuscript Listings*, Computer Systems Laboratory, Washington University, St. Louis, Mo., May 1967.

Reprinted from IEEE TRANSACTIONS
ON COMPUTERS
Volume C-19, Number 11, November, 1970
pp. 1009–1015