Data Compression via Textual Substitution

JAMES A. STORER AND THOMAS G. SZYMANSKI

Princeton University, Princeton, New Jersey

Abstract. A general model for data compression which includes most data compression systems in the literature as special cases is presented. Macro schemes are based on the principle of finding redundant strings or patterns and replacing them by pointers to a common copy. Different varieties of macro schemes may be defined by specifying the meaning of a pointer; that is, a pointer may indicate a substring of the compressed string, a substring of the original string, or a substring of some other string such as an external dictionary. Other varieties of macro schemes may be defined by restricting the type of overlapping or recursion that may be used. Trade-offs between different varieties of macro schemes, exact lower bounds on the amount of compression obtainable, and the complexity of encoding and decoding are discussed, as well as how the work of other authors relates to this model.

Categories and Subject Descriptors: E.4 [**Data**]: Coding and Information Theory—*data compaction and compression*; F.1.3 [**Computation by Abstract Devices**]: Complexity Classes—*reducibility and completeness*; F.2.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and Problems—*pattern matching*.

General Terms: Algorithms, Theory

Additional Key Words and Phrases: Textual substitution, macro expansion, dictionary, NP-completeness

1. *Introduction*

On-line secondary storage space is one of the most restricting resources in many modern computer installations, particularly in those employing multiuser time-sharing systems. Fast algorithms for compressing and restoring data files can do much to alleviate this problem. Some of the more popular data compression schemes described in the literature include *statistical encoding* techniques such as Huffman codes [8], which typically encode a block of source data as a variable-length string of bits determined by various statistical properties of the source data; *incremental encoding* methods (e.g., [21, 34]), which typically compress a file by recording only the difference between successive records; and textual substitution or *macro* encoding schemes (e.g., [6, 7, 12–14, 19, 20, 25–28, 30, 31, 33, 35, 37–39]), which factor out duplicate occurrences of data, replacing the repeated elements with some sort of special marker identifying the data to be replaced at that point. In addition, many ad-hoc methods for handling data with certain known characteristics appear in the literature.

This research was supported in part by the National Science Foundation under Grant MCS 74-21939 and in part by Bell Laboratories.

Authors' present addresses: J. A. Storer, Department of Computer Science, Brandeis University, Waltham, MA 02254; T. G. Szymanski, Bell Laboratories, 600 Mountain Avenue, Murray Hill, NJ 07974.

Journal of the Association for Computing Machinery, Vol. 29, No. 4, October 1982, pp. 928–951

This paper is devoted exclusively to the properties of the macro model for data compression. We study two major types of macro schemes, the types being differentiated by the location where the factored-out text is stored. Section 2 contains a discussion of our model along with some basic definitions, Sections 3 and 4 present our results for the two major types of schemes considered, and Section 5 examines the relative performance of the various compression schemes introduced in the preceding sections. To reduce the size of this paper, NP-completeness results are presented in [30]. However, the more important results of [30] are summarized here.

Before proceeding to the next section, we define the following notation:

(1) If s and s_i denote strings and $n \geq 1$ is an integer, $s_1 s_2$ denotes the concatenation of s_1 with s_2, $\prod_{i=1}^{n} s_i$ denotes $s_1 s_2 \cdots s_n$, and s^n denotes $\prod_{i=1}^{n} s$. s^0 denotes the empty string.

(2) We use the term *collection* to mean multiset.[1]

(3) If s is a string, $|s|$ denotes the length of s, and if s is a collection, $|s|$ denotes the number of elements in s (with each element being counted as many times as it appears in s).

(4) We extend the *min* function to strings by defining

$$\min\{s_1, s_2\} = \begin{cases} s_1 & \text{if } |s_1| \leq |s_2|, \\ s_2 & \text{otherwise.} \end{cases}$$

(5) For a real number h, $\lceil h \rceil$ denotes the least integer greater than or equal to h.

2. *The Model and Basic Definitions*

We shall treat the source data as a finite string over some alphabet. With *external macro schemes*, a source string is encoded as a pair of strings, a *dictionary* and a *skeleton*. The skeleton contains characters of the input alphabet interspersed with pointers to substrings of the dictionary. The dictionary is also allowed to contain pointers to substrings of the dictionary. The source string is recovered by substituting dictionary strings for pointers. With *internal macro schemes*, a string is compressed by replacing duplicate instances of substrings with pointers to other occurrences of the same substrings. The result is a single string of characters and pointers.

Throughout this paper let $p \geq 1$ denote the implementation-dependent size of a pointer.[2] If x is a string containing pointers, the *length* of x, denoted $|x|$, is defined to be the number of characters in x plus p times the number of pointers in x. We shall treat a pointer as an indivisible object which, in some unspecified fashion, uniquely and unambiguously identifies some string which is referred to as the *target* of that pointer. The way a pointer is written is not important; the only assumption we make is that it is always possible to determine by inspection of a pointer the length of its target.[3] For simplicity we shall write a pointer as a pair (n, m), where n indicates the position of the first character in the target,[4] m indicates the length of the target, and $|(n, m)|$ is the pointer size p. Without loss of generality it will always be assumed that $m > p$.

[1] A multiset is a set in which repetitions are allowed For example $\{a, a, b\}$ is a multiset.

[2] We assume that all pointers within a given string have a uniform size. (Variable-length pointers are considered in [30].) We also assume p to be an integer, although our results generalize to nonintegral pointer sizes

[3] It is not always necessary to make this assumption and, in fact, it can be useful to remove it See [30] for a discussion of this.

[4] n can be either an absolute location or a displacement. For example, with internal schemes, n could be the distance from the pointer to its target.

As an example of these ideas, let $p = 1$, and consider the string

$$w = aaBccDaacEaccFacac,$$

which might be encoded under the external macro model as

$$x = aacc\#(1, 2)B(3, 2)D(1, 3)E(2, 3)F(2, 2)(2, 2),$$

where # separates the dictionary from the skeleton. For convenience, we assume $|\#| = 0$. The compression achieved by the string x (i.e., the ratio $|x|/|w|$) is $\frac{14}{18}$. Using the internal macro model, w could be encoded as

$$y = aaBccD(1, 2)cEa(4, 2)Fac(13, 2),$$

achieving a compression of $\frac{15}{18}$.

Implementation considerations motivate us to describe a number of variations on our basic models. A scheme is *recursive* if a macro body (i.e., a string that is a target of a pointer) is allowed to itself contain pointers. Two pointers *overlap* if their targets overlap. Whether overlapping pointers are permitted in the external model depends highly on the implementation chosen for the dictionary.[5] An *original pointer* is one which denotes a substring of the original source string, whereas a *compressed pointer* denotes a substring of the compressed representation itself. The string y of the previous example contains compressed pointers. Using original pointers, we could encode w as

$$z = aaBccD(1, 2)cEa(4, 2)F(8, 2)(8, 2),$$

achieving a compression of $\frac{14}{18}$. Original pointers are more natural for one-pass decoding. Compressed pointers allow the recovery of portions of the source string without requiring the implicit decompression of the entire string. A *left* (*right*) *pointer* is one which denotes a substring occurring earlier (later) in the string. Considering the strings x, y, and z presented above, only x uses overlapping pointers, only z uses recursion, and none of these strings use right pointers. By using both left and right pointers it is possible to save additional space over the use of just one direction. For example, using both right and left pointers, the compressed forms y and z presented above could be replaced by

$$y = (5, 2)B(10, 2)DaacEaccF(6, 2)(6, 2),$$
$$z = (7, 2)B(12, 2)DaacE(8, 2)cF(8, 2)(8, 2),$$

achieving a compression of $\frac{14}{18}$ and $\frac{13}{18}$ respectively. We discuss recursion in relation to original pointers primarily to study the "power" of various methods. With original pointers, a pointer is recursive if all or part of the string it represents is represented by a pointer.

Cycles cannot occur in compressed forms with compressed pointers, but using original pointers, cycles can often make sense. For example, the compressed form $ab(5, 2)a(1, 3)$ uniquely determines the palindrome *abaaaaba* even though the two pointers in this compressed form comprise a cycle. Here the pointers (5, 2) and (1, 3) are a cycle in the sense that each points to a portion of the string represented by the other. An example of a degenerate cycle is given by the compressed form $a(1, n)$, which uniquely determines the string a^{n+1}. Schemes which allow recursion but not cycles are said to have *topological recursion*. From the above discussion it should be clear that topological recursion is not necessary for a compressed form to be uniquely

[5] Certain implementation considerations can lead to the placement of various restrictions on the kinds of overlapping permitted. Some of these restrictions are described in [30, 32].

decodable. However, it can be useful to consider topological recursion for three reasons. First, authors in the past (such as Lempel and Ziv) have assumed this. Second, study of such schemes leads to a deeper understanding of the power of original pointers. Third, topological recursion may model some practical considerations in the design of efficient original pointer compression methods.

The above discussion leads us to formally define four basic *macro schemes* and three types of *restrictions* which may be placed on any of these schemes. Throughout this paper, Σ denotes the underlying alphabet from which the data in question is constructed.

Definition 1. A *compressed form* of a string s using the *EPM* (*external pointer macro*) *scheme* is any string $t = s_0 \# s_1$ satisfying[6]

(1) s_0 and s_1 consist of characters from Σ and pointers to substrings of s_0.
(2) s can be obtained from s_1 by performing the following two steps:

   (a) Replace each pointer in s_1 with its target.
   (b) Repeat step A until s_1 contains no pointers.

Definition 2. A *compressed form* of a string s using the *CPM* (*compressed pointer macro*) *scheme* is any string t satisfying

(1) t consists of characters from Σ and pointers to substrings of t.
(2) s can be obtained from t by forming the string $t \# t$ and then decoding as with the EPM scheme.

Definition 3. A *compressed form* of a string s using the *OPM* (*original pointer macro*) *scheme* is any string t satisfying

(1) t consists of characters from Σ and pointers representing substrings of s.
(2) s can be obtained from t by replacing each pointer (n, m) by the sequence of pointers $(n, 1), (n + 1, 1), \ldots, (n + m - 1, 1)$ and then decoding as with the CPM scheme, with the stipulation that pointers are considered to have length 1.

Definition 4. A *compressed form* of a string s using the *OEPM* (*original external pointer macro*) *scheme* is any string $t = s_0 \# s_1$ satisfying

(1) t consists of characters from Σ and pointers.
(2) s_0 may be decoded using the OPM scheme to produce a string r. Furthermore, pointers in s_1 point to substrings of r.
(3) s may be obtained by replacing each pointer in s_1 with its target in r.

A *contraction* of a string s for pointer size p according to a given scheme is a shortest compressed form of s using that scheme with pointer size p. A contraction of a string s will be denoted by $\Delta(s)$.[7] We shall refer to the process of replacing a string r by a pointer as *factoring out* r and often refer to a string that is a target or potential target as a *factor*.

Definition 5. A CPM (OPM) pointer q_1 *depends on* pointer q_2 if the target of q_1 contains q_2 (all or part of the string represented by q_2) or if there is a pointer q_3 such that q_1 depends on q_3 and q_3 depends on q_2. A macro scheme is restricted to *no recursion* if dependent pointers are forbidden, and to *topological recursion* if no

[6] For convenience we assume throughout this paper that $|\#| = 0$.
[7] A string may have more than one minimal-length compressed form. For formal discussions we can always ensure that $\Delta(s)$ is unique by assuming a lexicographic ordering.

pointer may depend on itself; that is, it must be possible to sort topologically[8] the pointers of a compressed form according to their dependencies.

Definition 6. Two pointers *overlap* if their targets overlap and *strictly overlap* if their targets overlap but neither target is a substring of the other. A macro scheme is restricted to *no overlapping* if overlapping pointers are forbidden.

Definition 7. A CPM (OPM) pointer *q points to the left* if the leftmost character of its target is to the left of q (the leftmost character of the string represented by q). A *right pointer* is similarly defined. A macro scheme is restricted to *unidirectional pointers* if all pointers must point in the same direction (of course, with the EPM and OEPM schemes, this only applies to the external dictionary). As a special case of this, we can restrict a macro scheme to have only *left* or *right* pointers.

The different combinations of the four basic macro schemes we have defined and the recursion, overlapping, and pointer direction restrictions provide us with a large number of data compression methods. The combinations are sufficiently general to cover virtually all of the text substitution schemes proposed in the literature. Discussion of the utility and appropriateness of various restrictions to the models are deferred until later in the paper.

We have not discussed the concept of adding to pointers arguments which allow the specification of modifications to be made on a factor before it is substituted. Macro schemes with arguments generally have more power than ones without. Also, a few data compression methods presented in the literature require a macro scheme with arguments to model them, for example, the subsequence and supersequence compression methods discussed in [16]. Macro schemes with arguments will not be discussed in this paper, but it it should be noted that the macro model can be extended to allow this generality.

3. *The External Macro Model*

The external macro model views the collection of macro bodies as residing outside the remainder of the compressed string. This makes external schemes ideal for compressing collections of strings using a common dictionary. There are several reasons why it is more natural to treat all pointers as compressed pointers when discussing this model. First, authors in the past have used the EPM scheme and not the OEPM scheme (the authors mentioned in the introduction who used external schemes all considered variants of the EPM scheme). Second, it allows us to decompress arbitrary portions of the data without first having to produce the entire string. Third, compressed pointers often require less space than original pointers. For these reasons, we concentrate our attention in this section on the EPM model and then indicate how our results can be extended to the OEPM model. As we shall see in the next section, there are advantages to using original pointers over compressed pointers that justify consideration of the OEPM model. In many cases the extension of results to the OEPM scheme is trivial, since if recursion is forbidden, original and compressed pointers become equivalent in power. Similarly, if overlapping is forbidden, unidirectional and bidirectional pointers become equivalent.

THEOREM 1. *For all strings s, if only topological recursion is allowed, then (assuming s is compressible) both* $|\Delta_{EPM}(s)|$ *and* $|\Delta_{OEPM}(s)|$ *are*

(a) $\geq p \log_2(|s|/p) + 1.9p$.

[8] For a discussion of topological sorting, see [10].

(*b*) $\geq 3p \log_3(|s|/p) - 0.02p$ *when overlapping is forbidden.*
(*c*) $\geq 2(p|s|)^{1/2}$ *when recursion is forbidden.*
(*d*) $\geq 2(p|s|)^{1/2}$ *when both recursion and overlapping are forbidden.*
(*e*) (*a*)–(*d*) *hold even if pointers are required to be unidirectional.*

If nontopological recursion is allowed, then

(*f*) *The bounds of* (*a*)–(*e*) *hold for the EPM scheme.*

But

(*g*) $|\Delta_{OEPM}(s)| \geq 2p + 1$, *independently of what overlapping and pointer direction restrictions are made.*

Furthermore, all of the bounds in (*a*)–(*g*) *are tight; that is, each is attained for infinitely many strings s.*[9]

PROOF. For (a)–(e), since $|\Delta_{OEPM}(s)| \leq |\Delta_{EPM}(s)|$, it is sufficient to show that the OEPM scheme satisfies these bounds and the EPM scheme can attain them infinitely often.

First let us consider (a). We can assume that $s = a^{|s|}$ (for some a in Σ) because $|\Delta_{OEPM}(a^{|s|})| \leq |\Delta_{OEPM}(s)|$. It is easy to show that for some $p < k \leq 2p$ and n,

$$\Delta_{OEPM}(s) = a^k \prod_{i=1}^{n} q_i \# q_{n+1},$$

where q_i, $1 \leq i < n$, points to the string represented by everything to the left of it and q_{n+1} points to some substring of length $|s|$ in the dictionary. Since q_1 points to a string of k characters, we must have $k > p$, or else a shorter contraction of s could be produced. Similarly, if $k > 2p$, we could produce a shorter contraction by representing a^k by a^{k-p-1} followed by a pointer to this string. Thus we have

$$\begin{aligned}
|\Delta_{OEPM}(s)| &\geq k + np + p \\
&\geq p \log_2(|s|/k) + k + p \\
&\geq \min\{p\lceil \log_2(|s|/i)\rceil + i + p : p < i \leq 2p\} \\
&\geq p \log_2(|s|/p) + \min\{p(1 + h - \log_2 h) : 1 < h \leq 2\} \\
&> p \log_2(|s|/p) + 1.9p.
\end{aligned}$$

For any $p < i \leq 2p$ and n, the bound $\min\{p|\log_2(|s|/i)| + i + p : p < i \leq 2p\}$ is achieved exactly by the EPM scheme on the string $s = a^{i2^n}$.

We now consider (b). Again we can assume that $s = a^{|s|}$. Since overlapping is forbidden, the pointers of $\Delta_{OEPM}(s)$ can be divided into a sequence of sets $S_1, \ldots, S_m$ such that the pointers in S_i, $1 < i \leq m$, have targets whose compressed representation consists of pointers in some s_j, $j < i$; in fact, since we are concerned only with worst-case performance,[10] we can assume that $j = i - 1$. We can also assume that S_i, $1 \leq i \leq m$, contains at most three pointers. This is because for any $k \geq 4$ there are an i and j such that $2i + 3j \leq k \leq 2^i 3^j$, and so we can replace a set S_i of four or more pointers by a sequence of sets having at most three pointers, where the last set in the sequence will represent a string at least as long as that represented by the original sequence of four or more. Hence, for some $x \leq 2$ we can assume that for all $i > x$, S_i contains

[9] Actually, the bound in (a) is just an approximation for the expression $\min\{p|\log_2(|s|/i)| + i + p : p < i \leq 2p\}$, which is attained exactly infinitely often. Similarly, the bound in (b) is just an approximation for the expression $\min\{3p|\log_3(|s|/i)| + i : 2p < i \leq 4p\}$, which is attained exactly for infinitely many strings.

[10] It is not necessary to consider a compressed form of length x for a string a^y if there is a compressed form of length $< x$ ($\leq x$) for a string a^z where $z \geq y$ ($z > y$).

exactly three pointers. This is because we can assume that the sets of two come first and a sequence of three two-pointer sets can be replaced by two three-pointer sets. Given this, it can be assumed that for some $2p < k \leq 4p$, $0 \leq M \leq L \leq 1$, and n, $\Delta_{\text{OEPM}}(s)$ is of the form

$$a^k(q_0^2)^L(q_1^2)^M\left(\prod_{i=2}^{n} q_i^3\right)\#q_{n+1}^3,$$

where q_0 points to a^k; q_1 points to q_0^2; q_2 points to a^K, q_0^2, or q_1^2, depending on the values of L and M; and for $3 \leq i \leq n+1$, q_i points to q_{i-1}^3. Thus we have

$$\begin{aligned}
|\Delta_{\text{OEPM}}(s)| &\geq k + 3np + 2Lp + 2Mp \\
&\geq 3p\lceil \log_3(|s|/k2^L2^M)\rceil + k + 2Lp + 2Mp \\
&\geq 3p\lceil \log_3(|s|/k)\rceil + k + 3p(\tfrac{2}{3} - \log_3 2)(L+M) \\
&\geq 3p\lceil \log_3(|s|/k)\rceil + k \\
&\geq \min\{3p\lceil \log_3(|s|/i)\rceil + i : 2p < i \leq 4p\} \\
&\geq 3p\log_3(|s|/p) + \min\{p(h - 3\log_3 h) : 2 < h \leq 4\} \\
&> 3p\log_3(|s|/p) - 0.02p.
\end{aligned}$$

For any $2p < i \leq 4p$ and $n > 1$, the bound of $\min\{3p|\log_3(|s|/i)| + i : 2p < i \leq 4p\}$ is achieved using the EPM scheme with no overlapping on the string $s = a^{i3^n}$.

The proofs for (c) and (d) appear in [30]. All of the proofs of (a)–(d) make use of left pointers only, and so (e) follows. (f) follows trivially because compressed pointers cannot form cycles, and hence with the EPM scheme all recursion must be topological. For (g) we may again assume that $s = a^{|s|}$ for some a in Σ. If s is compressible using the EOPM scheme, then s must contain at least two pointers and at least one character. Hence $2p + 1$ is a lower bound. It is also tight, since $a(1, |s| - 1)\#(1, |s|)$ is a compressed form for $s = a^{|s|}$. □

For topological recursion, Theorem 1 says just what one would expect; there is an $\Omega(p \log|s|)$ lower bound on the size of a compressed string when recursion is allowed ($\log_2$ with overlapping and $\log_3$ without) and an $\Omega((p|s|)^{1/2})$ lower bound when recursion is not allowed. For nontopological recursion the bound of (g) may seem unnatural. Clearly we cannot represent a string of arbitrary length using a constant amount of space. The bound of (g) simply illustrates the need for the pointer size to be a function of the string size to model situations where pointers may indicate strings of arbitrary length.

It should also be pointed out that the bounds of Theorem 1 apply primarily to "pathological" strings; in practice, reducing the size of a file by a small constant factor may be very significant. However, much of the utility of Theorem 1 comes from the fact that it provides exact bounds which are needed in several of our NP-completeness[11] proofs.

The next theorem considers encoding algorithms for the EPM model. Wagner [35] presents a polynomial-time algorithm for compressing a string, assuming that the dictionary of macro bodies is given as input to the encoding algorithm. However, no mention is made as to how the selection of the best possible dictionary is accomplished. Several heuristic methods for constructing dictionaries have been presented in [20] and [25]. Neither of these guarantees optimal compression or even provides bounds on the compression that is obtainable. The reason for this gap in the literature is the NP-completeness of finding $\Delta_{\text{EPM}}(s)$.

[11] For a definition of NP-completeness and related terms, see [1]. All of our proofs show NP-completeness in the sense of Karp [9] (which implies that of [3]).

THEOREM 2. *Given a string s and an integer K, it is NP-complete to determine whether $|\Delta_{EPM}(s)| \leq K$ in any of the following situations:*

(*a*) *both recursion and overlapping allowed;*
(*b*) *recursion allowed, overlapping forbidden;*
(*c*) *recursion forbidden, overlapping allowed;*
(*d*) *both recursion and overlapping forbidden;*
(*e*) *unidirectional pointers and any of* (*a*)-(*d*).

Furthermore, the above are true regardless of whether p is part of the problem input or is constrained to be a fixed integer greater than 1. *In fact, we show* (*b*) *and* (*d*) *to be true even if $p = 1$.*

PROOF. It should be clear that no one part of the theorem directly implies any other. Thus several reductions are used. The reductions employed include the *node cover problem* [9], the *restricted node cover problem* [17], the *K-node cover problem* [30], and the *superstring problem* [4, 17]. As mentioned in the introduction, proofs of all NP-completeness results appear in [30]. However, we shall include the proof of (b) and (d) as a "sample proof."

Proof of (*b*) *and* (*d*) *for $p > 1$.* Let $G = (V = \{v_1, \ldots, v_n\}, E = \{e_1, \ldots, e_m\})$, K be an instance of the node cover problem, and let $p = p_0$. Let \$ be a special symbol, and let @ denote a new, distinct symbol each time it occurs. For v_i in V, let $V_i = \$v_i^{p-1}\$$, and for $e_i = (v_j, v_k)$ in E, let $E_i = \$v_j^{p-1}\$v_k^{p-1}\$$. Now let

$$s = \left(\prod_{i=1}^{p}\prod_{j=1}^{n} V_j@\right)\left(\prod_{i=1}^{m} E_i@\right).$$

We claim that G has a node cover of size $\leq K$ if and only if $|\Delta(s)| \leq |s| + K - m$.

First suppose that G has a node cover $X \subseteq V$ of size K. We shall construct a compressed form t for s (having length $|s| + K - m$), where t is of the form $s_0\#(\prod_{i=1}^{p}\prod_{j=1}^{n} \bar{V}_j@)(\prod_{i=1}^{m} \bar{E}_i@)$, where s_0 contains those V_i for which v_i is in X, and $\bar{V}_j$ is V_j if v_j is not in X and a pointer to v_j in s_0 if v_j is in X. If E_i is $\$v_j^{p-1}\$v_k^{p-1}\$$, then $\bar{E}_i$ is either $rv_k^{p-1}\$$ or $\$v_j^{p-1}q$, where r is a pointer to v_j in s_0 and q is a pointer to v_k in s_0. Since X is a node cover, this can always be done. If we now compute the length of t, $|s_0| = K(p+1)$, $|\prod_{i=1}^{p}\prod_{j=1}^{n} \bar{V}_j@| = |\prod_{i=1}^{p}\prod_{j=1}^{n} V_j@| - pK$, and $|\prod_{i=1}^{m} \bar{E}_i@| = |\prod_{i=1}^{m} E_i@| - m$. Hence $|\Delta(s)| \leq |t| = |s| + K(p+1) - pK - m = |s| + K - m$, as was to be shown.

Conversely, suppose that $|\Delta(s)| < |s| + K - m$. We shall show that G has a node cover of size at most K. First observe that since overlapping of pointer targets is forbidden, no pointer in $\Delta(s)$ can refer, for any strings x and y, to a string of the form $xv_i\$v_jy$, $x@y$, or $x@y$, since such a string can occur at most once in s and no gain can be achieved by factoring it out. Thus $\Delta(s)$ is of the form $s_0\#(\prod_{i=1}^{p}\prod_{j=1}^{n} \bar{V}_j@)(\prod_{i=1}^{m} \bar{E}_i@)$, where s_0 is a dictionary of macro bodies and the $\bar{V}_j$'s and $\bar{E}_i$'s are the shortest compressed forms of the V_j's and E_i's, respectively, using s_0. As mentioned earlier, without loss of generality we are assuming throughout this paper that every pointer references a string of length at least $p + 1$. Thus, since $|V_i| = p + 1$, we can infer that each $\bar{V}_i$ is either V_i itself or a pointer to an occurrence of v_i in s_0. Similarly, since $|E_i| = 2p + 1$, each $\bar{E}_i$ must either consist of E_i itself or else be a string of the form $rv_j^{p-1}\$$ or $\$v_j^{p-1}r$, where r is a pointer to some V_i in s_0. Now let L be the number of $\bar{E}_i$'s such that $\bar{E}_i = E_i$, that is, the number of $\bar{E}_i$'s that have not had a factor removed. Then $|\prod_{i=1}^{m} \bar{E}_i@| = |\prod_{i=1}^{m} E_i@| - (m - L)$, since removing a factor from an E_i saves one character. Let J be the number of V_i's in s_0. Then

$|s_0| = J(p+1)$ and $|\prod_{i=1}^{p}\prod_{j=1}^{n}\tilde{V}_j@| = |\prod_{i=1}^{p}\prod_{j=1}^{n}V_j@| - Jp$, because each v_i that is replaced by a pointer saves one character. Thus $|\Delta(s)| = J(p+1) + |s| - Jp - (m-L) = |s| + J + L - m$, and so $J + L \leq K$. We now claim that G has a node cover of size $J + L$ formed by taking the J nodes represented in s_0 and one node from each of the L edges not factored in $\Delta(s)$. Therefore G has a node cover of size K, as was to be shown.

Proof of (b) and (d) for $p = 1$. The following proof is similar to the original proof of this result [30] in that it employs a reduction to the node cover problem for degree three graphs. However, although the actual the actual construction is slightly longer, the proof of its correctness is simpler. This simplified proof is due to J. Gallant.

Let $G = (V = \{v_1, \ldots, v_n\}, E = \{e_1, \ldots, e_m\})$, K be an instance of the node cover problem for which all nodes have degree 3.[12] As in the proof for $p > 1$, let \$ be a special symbol, and let @ denote a new, distinct symbol each time it occurs. For v_i in V let

$$V_i = \$v_i\$ \qquad \text{and} \qquad W_i = \$^2v_i\$^2,$$

and for $e_i = (v_j, v_k)$ in E let

$$E_i = \$^2v_j\$^3v_k\$^2.$$

Now let

$$s = \left(\prod_{i=1}^{n}(V_i@W_i@)^2\right)\left(\prod_{i=1}^{m}E_i@\right).$$

The claim is that G has a node cover of size K if and only if $|\Delta(s)| \leq |s| + K - 14n$.

The first half of the proof argues that if X is a minimal node cover for G, then a compressed form for s can be constructed as follows:

(1) The two copies of each V_i go to a copy of V_i in the dictionary and two pointers in the skeleton.
(2) The two copies of each W_i go to two copies of \$ followed by a pointer to V_i followed by a \$ in the skeleton if v_i is not in X; otherwise the two copies of W_i go to W_i in the dictionary and two pointers in the skeleton.
(3) Each E_i representing $e_i = (v_j, v_k)$ goes to a pointer to $\$^2v_j\$^2$ followed by a pointer to $\$v_k\$$ followed by \$ if v_j is chosen to cover the edge; otherwise E_i goes to \$ followed by a pointer to $\$v_j\$$ followed by a pointer to $\$^2v_k\$^2$.

A compressed form for s as constructed above saves one character for each pair of V_i's for a total of n: four characters for each pair of W_i's when v_i is not in X, three characters for each pair of W_i's when v_i is in X for a total of $4n - K$, and six characters for each E_i for a total of $6m$. Since $m = \frac{3}{2}n$, this yields $|\Delta(s)| \leq |s| - n - 4n + K - 6m = |s| + K - 14n$.

The other half of the proof requires more work. Here it must be argued that the method of compressing s as described above is the best possible. This is done by considering several cases that rest heavily on the fact that all nodes in G have degree 3. The degree-3 restriction makes it unprofitable to factor out many strings that might otherwise be factored if nodes with large degrees were present. We leave the details of this half of the proof to the reader. Note that the above reduction is for the case where recursion is forbidden. If recursion is allowed, the same construction may be used, except that one copy of W_i should be used instead of two. □

[12] Using a result from [5], it is easy to show this restriction of the node cover problem to be NP-complete; see [17].

In [30], cases (b)–(e) of Theorem 2 are shown to hold for the EOPM scheme. Case (a) is shown for the problem of compressing collections, but the single string problem remains open at the time of the writing of this paper. In addition, in [30] it is conjectured that all of (a)–(e) of Theorem 2 can be shown for $p = 1$.

Throughout this paper we assume that the alphabets over which strings are written are unbounded in size. However, results concerning lower bounds on encoding complexity are stronger if they apply to the case where all strings are assumed to be written over some fixed size alphabet, and, although unbounded size alphabets model many practical situations (such as when entries in a system dictionary are taken to be the basic characters), there are certainly many situations in which it is more realistic to consider strings to be written over some fixed finite alphabet. Thus it is worthwhile to consider the complexity of compressing strings when it is assumed that all strings are written over a fixed size alphabet. Since our motivation for doing this is to model practical situations, when discussing fixed size alphabets we also require that pointers of a given size can only encode a finite amount of information. This requirement is met by stipulating that the pointer size p be dependent on the string being processed. Because complexity results concerning fixed size alphabets are more technical, we shall only state a few sample theorems to indicate the flavor of these results and only for the case when both recursion and overlapping are forbidden. Suppose we allow pointers to be able to indicate any substring of the source. Then a pointer's length must be some implementation-dependent multiple of the logarithm of the string length.

THEOREM 3. *If recursion and overlapping are forbidden, then, giving a string s over any alphabet with at least three symbols, an integer K, and a real $h > 0$, it is NP-complete to determine whether s has an EPM compressed form t satisfying $|t| \leq K$ when the pointer size is $\lceil h \log_2 |t| \rceil$.*

Two other natural ways to determine pointer size are either to require that the information content of a pointer be sufficient to distinguish all the pointers in an encoding or to require that a pointer be able to identify any substring of the dictionary. To this end, if t is an encoding of some string using the EPM scheme, let $\delta(t)$ be the number of distinct pointers in t, and let $d(t)$ be the dictionary portion of t.

THEOREM 4. *If recursion and overlapping are forbidden, then, given a string s over any alphabet with at least three symbols, an integer K, and a real $h \geq 1$, it is NP-complete to determine whether s has an EPM compressed form t satisfying $|t| \leq K$ in the following situations:*

(a) p is $\lceil h \log_2 \delta(t) \rceil$.
(b) p is $\lceil h \log_2 |d(t)| \rceil$.

In [30], results similar to the above are shown for other combinations of restrictions. Although Theorems 3 and 4 apply only for alphabets of size 3 or greater, we conjecture that these results can be strengthened to apply for two-symbol alphabets.

The proofs of the last two theorems involve an extra level of complexity over the corresponding proofs for the unbounded alphabet cases, because when one attempts to embed, say, an NP-complete graph problem in a data compression problem, one is forced to encode nodes of the graph as strings. Care must be taken to ensure that the integrity of these strings is maintained during compression.

As was indicated at the start of this section, external macro schemes are useful for compressing collections of strings. Many of the NP-completeness results of this

section can be strengthened when applied to collections. For example, some results concerning bounded-size alphabets, when extended to collections, apply for two-symbol alphabets. Storer [30] also contains results concerning limitations on the size of strings in a collection and factors in compressed forms.

4. *The Internal Macro Model*

In the internal macro model it is rather unnatural to forbid the use of recursion or overlapping. We shall therefore concentrate on the four combinations provided by choosing between compressed and original pointers and choosing between unidirectional and bidirectional pointers. In addition, we consider the topological recursion restriction, not because we are necessarily claiming it to be a natural restriction for the OPM scheme, but because studying topological recursion appears to lend insight into the relative power of compressed and original pointers. As done in the last section, we first start with some performance bounds.

THEOREM 5. *For all strings s (assuming s is compressible),*

(a) $|\Delta_{OPM}(s)| \geq p + 1$;

(b) *for topological recursion,*[13] *both* $|\Delta_{CPM}(s)|$ *and* $|\Delta_{OPM}(s)|$ *are* $\geq p \log_2(|s|/p) + 0.9p$.

Furthermore, the above bounds are tight[14] *and hold regardless of whether unidirectional or bidirectional pointers are used.*

PROOF. Similar to that of Theorem 1. □

The next theorem deals with the relative power of compressed and original pointers.

THEOREM 6. *For any string s, $|\Delta_{OPM}(s)| \leq |\Delta_{CPM}(s)|$, independently of what restrictions are made (provided the same restrictions are used for both). Furthermore, for any real $h > 0$:*

(a) *Using any alphabet of size ≥ 1, there are infinitely many strings s for which*

$$\frac{|\Delta_{OPM}(s)|}{|\Delta_{CPM}(s)|} < h.$$

(b) *For topological recursion, using any alphabet of size ≥ 2, there are infinitely many strings for which*

$$\frac{|\Delta_{OPM}(s)|}{|\Delta_{CPM}(s)|} < \frac{1}{3} + h.$$

PROOF. Since a compressed pointer may always be converted to an original pointer, it follows that for any string s, $|\Delta_{OPM}(s)| \leq |\Delta_{CPM}(s)|$ independently of what restrictions are made. (a) follows trivially from Theorem 5, since for the string $s = a^n$, $|\Delta_{OPM}(s)| = p + 1$ but $|\Delta_{CPM}(s)|$ is $o(p \log_2 n)$.

Let us now consider (b). For n a multiple of p define

$$s_n = a^n b^n \prod_{i=1}^{n/p} (a^{ip} b^{n-(i-1)p}).$$

[13] Remember that with compressed pointers, all recursion must be topological.

[14] Similarly to Theorem 1(a), the bound of B is just an approximation for the expression $\min\{p \cdot |\log_2(|s|/i)| + i : p < i \leq 2p\}$, and it is this value that is achieved exactly by infinitely many strings.

Using the OPM scheme, s_n can be represented by the compressed form

$$t \prod_{i=1}^{n/p} (n - ip + 1, n + p),$$

where t is the best compressed representation of $a^n b^n$. Since $|t| = O(p \log_2 n)$, we have $|\Delta_{\text{OPM}}(s_n)| \leq p(n/p) + O(p \log_2 n) = n + O(p \log_2 n)$. On the other hand, if we attempt to factor s_n using the CPM scheme, a shortest compressed form is

$$a^n b^n \prod_{i=1}^{n/p} (n - ip + 1, n + p),$$

that is, the leading factor of $a^n b^n$ is preserved intact. Here the n/p pointers to the right of $a^n b^n$ that point into $a^n b^n$ "chop up" $a^n b^n$ so that it cannot be factored with compressed pointers; that is, if $a^n b^n$ were factored, then at least some of the pointers to the right of $a^n b^n$ would be pointing to "part of a pointer" which is not allowed for compressed pointers. Through the analysis of several cases, it can be shown that the above compressed form is the best possible, and thus $|\Delta_{\text{CPM}}(s_n)| = 3n$. Hence, for any real $h > 0$ we have, for sufficiently large n,

$$\frac{|\Delta_{\text{OPM}}(s_n)|}{|\Delta_{\text{CPM}}(s_n)|} = \frac{n + O(p \log_2 n)}{3n} \leq \frac{1}{3} + h. \qquad \square$$

We do not yet know whether the bound in (b) is the best possible. Also, it should be noted that although $|\Delta_{\text{OPM}}(s)| \leq |\Delta_{\text{CPM}}(s)|$, in principle it is possible for a compressed pointer to require less space than its corresponding original pointer since, for a given string s, compressed pointers may point to smaller strings.

We now consider performance bounds concerning pointer direction. First, it is obvious that using either the CPM or OPM schemes, for any string s,

(1) $|\Delta_{\text{L}}(s)| = |\Delta_{\text{R}}(t)|$ where t is the reverse of s;
(2) $|\Delta_{\text{L}}(s)|, |\Delta_{\text{R}}(s)| \geq |\Delta_{\text{UD}}(s)| \geq |\Delta(s)|$.

In view of this, we shall not bother to state "dual" theorems, that is, theorems that may be obtained from a previous theorem by changing all occurrences of $\Delta_{\text{L}}(s)$ to $\Delta_{\text{R}}(s)$, etc.

Before proceeding we present a short technical lemma that allows us to relate results concerning left versus right pointers to results concerning unidirectional versus bidirectional pointers.

LEMMA 1. *Using any macro scheme, for a given alphabet Σ, if there are infinitely many strings s over Σ for which $|\Delta_{\text{L}}(s)|/|\Delta_{\text{R}}(s)| < h$, then there are infinitely many strings s' over an alphabet Σ', where $|\Sigma'| = 2|\Sigma|$, for which*

$$\frac{|\Delta_{BD}(s')|}{|\Delta_{UD}(s')|} < \frac{2h}{1 + h}.$$

PROOF. Given a string s over an alphabet $\Sigma = \{a_1, \ldots, a_n\}$ for which $|\Delta_{\text{L}}(s)|/|\Delta_{\text{R}}(s)| < h$, let $\Sigma' = \{a'_1, \ldots, a'_n\}$ be new symbols not in Σ, and let $s' = st$, where t is the reverse of the string obtained by replacing each symbol a_i in s by a'_i. It is easy to check that

$$\frac{|\Delta_{\text{BD}}(s')|}{|\Delta_{\text{UD}}(s')|} \leq \frac{2 \min\{|\Delta_{\text{L}}(s)|, |\Delta_{\text{R}}(s)|\}}{|\Delta_{\text{L}}(s)| + |\Delta_{\text{R}}(s)|} < \frac{2h}{1 + h},$$

and that s' is written over an alphabet of size $2|\Sigma|$. $\square$

THEOREM 7. *Let OPM/TR denote the OPM scheme restricted to topological recursion. For any real $h > 0$:*

(a) There are infinitely many strings s for which

$$\frac{|\Delta_{CPM/R}(s)|}{|\Delta_{CPM/L}(s)|} < \frac{1}{2} + h.$$

(b) Using any alphabet of size 2 or more, there are infinitely many strings s for which

$$\frac{|\Delta_{OPM/R}(s)|}{|\Delta_{OPM/L}(s)|} < max\left\{\frac{1}{3}, \frac{p}{2p+2}\right\} + h$$

and

$$\frac{|\Delta_{OPM/TR/R}(s)|}{|\Delta_{OPM/TR/L}(s)|} < \frac{1}{3} + h.$$

PROOF

(a) For any integer $n > 1$, let

$$s_n = \prod_{i=1}^{n-1} \left(\left(\prod_{j=1}^{n-1} t_{i,j} \right) b_i \right),$$

where $t_{i,j} = \prod_{k=j}^{j+i} a_k^p$. Using right pointers, each $t_{i,j}$ except $t_{n-1,1}$ can be replaced by a pointer into $t_{n-1,1}$. Hence we have

$$\begin{aligned} |\Delta_{CPM/R}(s_n)| &\leq \left(\sum_{i=2}^{n-1} ip \right) + np + n - 1 \\ &= \tfrac{1}{2}pn^2 + O(pn). \end{aligned}$$

Using left pointers, the target of a pointer must be a substring of the compressed form of $t_{i,j}$ for some i and j. This is because for each $t_{i,j}$ and $t_{i,j+1}$, if a is the last character of $t_{i,j}$ and b the first character of $t_{i,j+1}$, the string ab occurs nowhere else in s_n (in addition, the b_i's separate $s_{i,n-i}$ and $s_{i+1,1}$). Thus, since $t_{i,j}$ cannot be a substring of $t_{x,y}$ if $i < x$ or $i = x \;\&\; y < j$, we have

$$\begin{aligned} |\Delta_{CPM/L}(s_n)| &\geq \left(\sum_{i=1}^{n-1} 2ip \right) + n - 1 \\ &= pn^2 + O(pn), \end{aligned}$$

and hence for sufficiently large n,

$$\frac{|\Delta_{CPM/R}(s_n)|}{|\Delta_{CPM/L}(s_n)|} = \frac{pn^2/2 + O(pn)}{pn^2 + O(pn)} < \frac{1}{2} + h.$$

(b) Let us first consider topological recursion. For $n > 0$ let

$$s_n = \prod_{i=1}^{n+1} a^{ip} b^{ip}.$$

Since S_n can be factored by replacing each string $a^{ip}b^{ip}$, $1 \leq i \leq n$, by a pointer into the string $a^{(n+1)p}b^{(n+1)p}$, it follows that

$$|\Delta_{OPM/R}(s_n)| \leq pn + O(p \log_2(pn)).$$

The key observation for calculating $|\Delta_{\text{OPM/L}}(s_n)|$ is to note that the largest possible factor to the left of $a^{ip}b^{ip}$, $1 < i \leq n+1$, is $a^{(i-1)p}b^{(i-1)p}$. From this reasoning it follows that $|\Delta_{\text{OPM/L}}(s_n)| = 3pn + 2p$. Hence,

$$\frac{|\Delta_{\text{OPM/R}}(s_n)|}{|\Delta_{\text{OPM/L}}(s_n)|} < \frac{1}{3} + h$$

for sufficiently large n.

The construction for nontopological recursion is the same, except that we now have the option of writing a^ib^j as aq_1bq_2, where q_1 and q_2 are the appropriate pointers. □

COROLLARY 7.1. *The bounds* $\frac{1}{2} + h$, $\max\{\frac{1}{3}, p/(2p+2)\} + h$, *and* $\frac{1}{3} + h$ *stated in Theorem* 7 *are for left versus right pointers. If, instead, unidirectional and bidirectional pointers are compared, then the bounds* $\frac{2}{3} + h$, $\max\{\frac{1}{2}, 2p/(3p+2)\} + h$, *and* $\frac{1}{2} + h$ *follow. Furthermore, the alphabet sizes needed are at most double the sizes stated in Theorem* 7.

PROOF. Follows directly from Lemma 1. □

We do not yet know whether the bounds of Theorem 7 are tight. One technique for deriving lower bounds on ratios concerning pointer directions is to consider the *overlapping content* of a compressed form. We have not been able to use this to prove or disprove that the bounds of Theorem 7 are tight, but the concept of overlapping content does lead to some interesting ideas. We digress and consider this in relation to the CPM scheme.

Definition 8. The *overlapping content* of a pair of pointers q_i and q_j for pointer size p is given by

$$\text{OVCON}(q_i, q_j) = \begin{cases} 0 & \text{if } q_i \text{ and } q_j \text{ do not strictly overlap,} \\ k & \text{if } q_i \text{ and } q_j \text{ strictly overlap by } < p \text{ characters,} \\ p & \text{if } q_i \text{ and } q_j \text{ strictly overlap by } \geq p \text{ characters,} \end{cases}$$

and the overlapping content of a compressed form t is given by

$$\text{OVCON}(t) = \sum_{(q_i,q_j)\in t} \text{OVCON}(q_i, q_j).$$

THEOREM 8. *For any string* s,

$$|\Delta_{CPM/L}(s)| \leq |\Delta_{CPM}(s)| + OVCON(\Delta_{CPM}(s)).$$

PROOF. For a string s, given $\Delta_{\text{CPM}}(s)$ (or any compressed form for s), all strict overlapping can be removed to obtain a new compressed form t for s as follows:

while there is a string uvw such that uv is the target of some pointer q_i and vw is the target of some pointer q_j **do**

Replace q_i by q_uq_v where

$$q_u = \begin{cases} u & \text{if } |u| \leq p, \\ \text{a pointer to } u & \text{otherwise,} \end{cases}$$

$$q_v = \begin{cases} v & \text{if } |v| \leq p, \\ \text{a pointer to } v & \text{otherwise.} \end{cases}$$

The reader can check that $|t| \leq |\Delta_{\text{CPM}}(s)| + \text{OVCON}(\Delta_{\text{CPM}}(s))$.

We now describe how to convert t to a left pointer CPM compressed form for s without increasing its size.

Sort the pointers in t topologically as $q_1, \ldots, q_n$ so that q_i, $1 \le i \le n$, is not pointed to (directly or indirectly) by any q_j, $i < j \le n$. Since we are allowing substring overlapping, we assume that if q_i points to uvw and q_j points to v, then $i \le j$. Now, starting with q_1, do the following to each pointer q_i. If q_i points to the left, do nothing. Otherwise, if q_i points to a string w to its right, swap q_i with w and adjust all other pointers accordingly; that is, all pointers that point to some substring of w must be changed to point to this substring in the new position of w.

The key point in verifying that the above procedure works is that at the ith stage, pointers q_1 through q_{i-1} are not disturbed. □

Since $\mathrm{OVCON}(\Delta_{\mathrm{CPM}}(s))$ can be as large as $O(|s|^2)$, it is possible for Theorem 8 to yield very poor bounds. However, this is not always the case, as seen by the following example.

Example 1. A CPM compressed form t has *simple overlapping* if each pointer in t strictly overlaps with at most one pointer to its right and one to its left. Let CPM/SO denote the CPM scheme restricted to simple overlapping. A CPM/SO compressed form with n pointers has at most $n - 1$ strict overlaps. Thus, since the construction of Theorem 8 preserves simple overlapping, for all strings s,

$$\begin{aligned} |\Delta_{\mathrm{CPM/L/SO}}(s)| &\le |\Delta_{\mathrm{CPM/SO}}(s)| + \mathrm{OVCON}(\Delta_{\mathrm{CPM/SO}}(s)) \\ &< np + (n-1)p \\ &< 2|\Delta_{\mathrm{CPM/SO}}(s)|. \end{aligned}$$

□

We now turn our attention to the complexity of optimally compressing strings using the internal macro model. Unfortunately, like the EPM scheme, optimal encoding for the CPM scheme appears to be intractable.

THEOREM 9. *Given a string s and an integer K, it is NP-complete to determine whether $|\Delta_{CPM}(s)| \le K$ even when any combination of the restrictions to unidirectional pointers, no recursion, and no overlapping is made. Furthermore, this is true regardless of whether the pointer size p is part of the problem input or is constrained to be a fixed integer greater than* 1.

PROOF. First let us assume that overlapping is forbidden. In this case we can use a construction similar to that used for Theorem 1(b) and (d). Let $G = (V = \{v_1, \ldots, v_n\}, E = \{e_1, \ldots, e_m\})$, K be an instance of the node cover problem and, as in the proof of Theorem 1, let \$ be a special symbol, and let @ denote a new, distinct symbol each time it occurs. Now, for $e_i = (v_j, v_k)$ in E, let $E_i = \$v_j^{p-1}\$v_k^{p-1}\$$, and let $s = \prod_{i=1}^{m} E_i@$.

We claim that G has a node cover of size K if and only if $|\Delta(s)| \le |s| + K - m$. If G has a node cover X of size K, then for each node in X we can associate an E_i in s (which will not get factored), and for all remaining $m - K$ E_i's we can replace exactly one string of the form $\$v_j\$$ by a pointer. Thus $|\Delta(s)| \le |s| + K - m$. The proof of the converse is very similar to that used in Theorem 1(b) and (d), and we omit the details. As with Theorem 1, the above construction works independently of whether recursion is allowed (since the largest factor has length $p + 1$). In addition, the left, right, or unidirectional pointer restrictions cause no problem. For example, if we are restricted to left pointers, we consider nodes in X one at a time and associate each with the leftmost "unassociated" E_i containing it.

When overlapping is allowed, we can use the same construction as above, except that we let G, K be an instance of the 1-*node cover problem* which is defined as: Given a graph G and integer K, is there a set of K or fewer edges in G such that every edge in G is adjacent to at least one of these edges? The 1-node cover problem can be shown to be NP-complete as follows. For $G = (V = \{v_1 \cdots v_n\}, E =$

$\{e_1 \cdots e_m\}$), K an instance of the node cover problem, construct the graph $G' = (V', E')$, where V' is V together with the new nodes x_i and y_i, $1 \le i = K$, and E' is E together with the new edges (v_i, x_j) and (x_j, y_j), $1 \le i \le n$ and $1 \le j \le k$. Then G has a node cover of size K if and only if G' has a node cover of size K.

The only remaining case is the CPM scheme (with or without recursion) with one of the pointer direction restrictions. This requires a separate construction which appears in [30] for the case $p \ge 5$. □

The situation for the OPM scheme is much better. Although, at the time of the writing of this paper, the status of the encoding complexity of the OPM scheme with bidirectional pointers remains open,[15] we shall show that the unidirectional case can be done in linear time.[16] Lempel and Ziv in [12] (and also in [39]) have developed a data compression algorithm that falls within the framework of our OPM scheme restricted to left pointers and topological recursion. (As we shall see from the proof of Theorem 11, a linear-time encoding algorithm for left pointers implies a linear-time encoding algorithm for unidirectional pointers.) Rodeh et al. [24] have presented a linear-time implementation of the Lempel–Ziv algorithm using the techniques of [15].[17] Their implementation can most simply be described as a one-pass greedy algorithm. At each step the longest possible prefix of the remaining input that matches some substring of the previously read input is removed from the input and replaced with a pointer. For example, if we have already processed *ababc* and the rest of the input is *abcd*, then we would output the pointer (3, 3) and delete the next three characters of the input. The Lempel–Ziv algorithm is asymptotically optimal for ergodic sources as the length of the source string tends to infinity; however, for individual finite strings the compression achieved can be far from optimal.

THEOREM 10. *Let $LZ(s)$ denote the compressed form of s obtained by applying the Lempel–Ziv algorithm. Then for any string s,*

$$\text{if } p = 1, \quad \text{then } |LZ(s)| = |\Delta_{OPM/TR/L}(s)|,$$

$$\text{if } p > 1, \quad \text{then } \frac{p}{2p-1} < \frac{|\Delta_{OPM/TR/L}(s)|}{|LZ(s)|} \le 1.$$

Furthermore, for any real number $h > 0$ it is possible to construct a string s over a two-symbol alphabet such that $|\Delta_{OPM/TR/L}(s)|/|\Delta_{LZ}(s)| \le (p+1)/2p + h$.

PROOF. Without loss of generality we can assume that in any minimal-length compressed form, any substring that is represented by a pointer to an earlier occurrence is as long as possible; that is, if $s_m \cdots s_n$ is represented by a pointer, then $s_m \cdots s_{n+1}$ is not a substring of $s_1 \cdots s_{m-1}$. Otherwise we could obtain an equivalent compressed form of the same or shorter length by changing the pointer to represent $s_m \cdots s_{n+1}$ and then either deleting a character (if the pointer was originally followed by a character) or changing the following pointer (if the pointer was originally followed by another pointer).

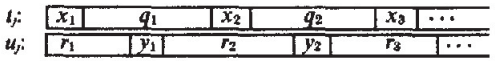
Let s be any string, and consider $t = \Delta_{OPM/TR/L}(s)$ and $u = \Delta_{LZ}(s)$. Form the finest partition of t and u into segments $t = t_1 \cdots t_m$ and $u = u_1 \cdots u_m$ such that for $1 \le$

[15] Recently, it has been shown by J. Gallant (in his Ph D. dissertation, "String Compression Algorithms," Princeton University) that this problem is, in fact, NP-complete

[16] Note that as with the CPM scheme, encoding for the OPM scheme with either or both of the recursion and overlapping restrictions (with unidirectional or bidirectional pointers) is NP-complete. A proof of this may be found in [30]

[17] In addition to [15], the interested reader should refer to [2, 11, 18, 22, 29, 36]. Also, a good introduction to the above work is contained in [1]

FIGURE 1



$j \leq m$, t_j and u_j represent the same substring of s. In order to establish the bounds quoted in this theorem, it is sufficient to show that

$$|t_1| = |u_1| = 1,$$

$$\frac{p}{2p-1} \leq \frac{|t_j|}{|u_j|} \leq 1 \qquad \text{for} \quad j > 1.$$

By definition of the Lempel–Ziv algorithm, it is impossible for some t_j to begin with a pointer while u_j begins with a character. We therefore have one of the following cases:

(1) t_j and u_j both consist of a single character.
(2) t_j and u_j both consist of a single pointer (which represent identical strings by the optimality principle stated at the beginning of the proof).
(3) t_j begins with a character, and u_j begins with a pointer.

In the first two cases, $|t_j| = |u_j| = 1$, and their ratio falls within the desired bounds. We must therefore establish the bounds for case (3). Let us write

$$t_j = x_1 q_1 x_2 q_2 \cdots x_n q_n x_{n+1},$$
$$u_j = r_1 y_1 r_2 y_2 \cdots r_m y_m,$$

where each of the x_i's and y_i's is a string of zero or more characters and the q_i's and r_i's are pointers.

A key observation is that any substring of s that is represented by characters (as opposed to pointers) in either t_j or u_j must be represented by a pointer in the other. This is true because of our definition of t_j and u_j in terms of a finest possible partition of t and u. Figure 1 suggests the structure of that portion of s represented by t_j and u_j. Notice that for each i, $1 \leq i \leq n$, q_i represents at least the last character represented by r_i, all of y_i, and at least the first character represented by r_{i+1}. Also, for $2 \leq i \leq m$, r_i represents at least the last character represented by q_{i-1}, all of x_i, and at least the first character represented by q_i. To verify the above facts, depicted in Figure 1, it is sufficient to observe that except at the end, if q_i starts within r_i, then q_i must go beyond the end of r_i, since if q_i ended earlier, then q_i would not be as long as possible (as we assumed at the start of the proof), and if q_i ended at the same place, we would not have the finest possible partition. Similarly, if r_i starts within a q_i, then r_i must go beyond the end of q_i, since to end earlier would imply a violation of the Lempel–Ziv greedy rule, and to end in the same place would violate the finest partition.

Let us summarize some important observations about Figure 1:

(1) Either $m = n$ or else $m = n + 1$.
(2) For $1 \leq i \leq n + 1$, $|x_i| \leq p$, or else we could replace x_i with a pointer to some earlier occurrence in s, thus reducing the length of t by at least one in contradiction to our definition of t as a compressed form of minimal length.
(3) For $1 \leq i < m$, $|y_i| \leq p - 1$, or else the Lempel–Ziv algorithm would have used a pointer instead of y_i.
(4) $|y_m| \leq p$.

A number of cases now arise.

Case 1. Suppose that $n = m$. Since t_j contains exactly n pointers and at least one character from x_1, we have $|t_j| \geq np + 1$. Now consider u_j, which also has exactly n pointers. Since each of the y_i (except possibly y_n) has no more than $p - 1$ characters,

$$|u_j| \leq np + (n-1)(p-1) + p = n(2p-1) + 1.$$

Thus

$$\frac{|t_j|}{|u_j|} \geq \frac{np+1}{n(2p-1)+1} \geq \frac{p}{2p-1}.$$

Case 2. Suppose that $m = n + 1$ and x_{n+1} is the empty string. Thus both t_j and u_j end in a pointer. It is not hard to see that y_{m-1} must also be empty, or else the Lempel–Ziv algorithm would have replaced the string represented by $y_{m-1}r_m$ by a single pointer. Thus u_j contains exactly $n + 1$ pointers, $|y_k| \leq p - 1$, for $1 \leq k \leq n - 1$, and $|y_n| = |y_{n+1}| = 0$. Hence,

$$|u_j| = (n+1)p + (n-1)(p-1) = (2p-1)n + 1.$$

Once again we have

$$\frac{|t_j|}{|u_j|} \geq \frac{np+1}{n(2p-1)+1} \geq \frac{p}{2p-1}.$$

Case 3. Suppose that $m = n + 1$ and x_{n+1} is not empty. By our definition of t and u in terms of a finest possible partition, it must be the case that y_{n+1} is the empty string. Also, since the string represented by q_n extends at least $(p + 1) - |x_{n+1}|$ characters past y_n, it must be that $|y_n| < |x_{n+1}|$; otherwise, the presence of q_n implies that the Lempel–Ziv algorithm must place a pointer directly after r_n (i.e., $|y_n| = 0$). Thus we have

$$\frac{|t_j|}{|u_j|} \geq \frac{np + |x_{n+1}| + 1}{n(2p-1) + |y_n| + 1} > \frac{np+1}{n(2p-1)+1} \geq \frac{p}{2p-1}.$$

In all of the above cases we have shown that $|t_j|/|u_j| \geq p/(2p-1)$. Since we are using left pointers, it must be that t_1 and u_1 contain no pointers (and so $|t_1| = |u_1|$). Thus $|t| = |u|$ for $p = 1$, and $|t|/|u| > p/(2p-1)$ for $p > 1$.

For any $p > 1$, using only a two-symbol alphabet, we can approach the lower bound of $p/(2p-1)$ as follows. For $k \geq 0$, let $n = (p+1)2^k - 1$ and

$$s_n = ab^{n+1}ab^{n+1}\prod_{i=0}^{n-p}(ab^{n-i}ab^{n-i}).$$

It is easy to check that

$$\begin{aligned}\frac{|\Delta_{\text{OPM/TR/L}}(s_n)|}{|\Delta_{\text{LZ}}(s_n)|} &= \frac{(k+2p+1)+(n-p)(p+1)}{(k+2p+1)+(n-p)2p} \\ &= \frac{n(p+1)+O(p\log_2(n))}{n(2p)+O(p\log_2(n))} \\ &\to \frac{p+1}{2p} \quad \text{as} \quad n \to \infty.\end{aligned}$$

For $p = 1$, $p/(2p-1) = (p+1)/2p = 1$, and as p gets large, both quantities converge to $\frac{1}{2}$. Nevertheless, for $p > 1$, $p/(2p-1)$ is strictly less than $(p+1)/2p$, and so we are left with a small "gap." At the time of the writing of this paper, this gap has not been resolved. □

Theorem 10 shows that as p gets large, the worst-case performance of the Lempel–Ziv algorithm does not compare favorably with that of $\Delta_{OPM/TR/L}$. The next theorem shows that if we compare the performance of the Lempel–Ziv algorithm with that of $\Delta_{OPM/L}$, the disparity becomes even greater.

THEOREM 11. *For all strings and any pointer size p,*

$$0 < \frac{|\Delta_{OPM/L}(s)|}{|\Delta_{LZ}(s)|} \leq 1,$$

and the above bounds are tight.

PROOF. This is a direct consequence of Theorems 5 and 10. □

Although the Lempel–Ziv algorithm is not optimal for the OPM/L scheme even when it is restricted to topological recursion, the next theorem shows that a linear-time algorithm does exist for optimally compressing strings using the OPM scheme restricted to unidirectional pointers of any size (independent of whether topological recursion is used). In view of the number of NP-completeness results presented thus far, this is a pleasing result, especially since the OPM scheme has many practical applications.

THEOREM 12. *For any string s, $\Delta_{OPM/UD}(s)$ can be constructed in linear time (on a RAM).*

PROOF. Given a string $s = s_1 \cdots s_n$, $\Delta_{OPM/L}(s)$ may be computed by performing the following steps (note that SHORT[ ] and MATCH[ ] are arrays of strings):

A: Let MATCH[k], $1 \leq k \leq n$, be the longest string $s_i \cdots s_j$ such that $i < k$ and $s_i \cdots s_j = s_k \cdots s_{k+j-i}$. Also, let q_k denote a pointer to MATCH[k].

B: SHORT[$n + 1$] = ⟨empty string⟩

C: **do** $i = n - 1$ **to** 1 **by** -1;
  if MATCH[i] = ⟨empty string⟩ **then** SHORT[i] = s_iSHORT[$i + 1$]
  else SHORT[i] = min{s_iSHORT[$i + 1$], q_iSHORT[$i + |$MATCH[i]$|$]}

D: $\Delta_{OPM/L}(s)$ = SHORT[1]

The algorithm is a dynamic programming algorithm which utilizes the optimality principle stated at the beginning of the proof of Theorem 10. Each string SHORT[i] computed by the algorithm is the shortest compressed form for $s_i \cdots s_n$, given that $s_1 \cdots s_{i-1}$ is available as a "dictionary." By using the appropriate data structures, step A can be performed in linear time using a slight generalization of the algorithm described in [23]. To perform step C in linear time, we note that the array SHORT can be represented by storing at SHORT[i]s_i (or q_i) followed by a pointer to SHORT[$i + 1$] (or SHORT[$i + |$MATCH[i]$|$]). In step D we can easily write out SHORT[1] in linear time by following the sequence of pointers through the array SHORT. Hence the entire algorithm to compute $\Delta_{OPM/L}(s)$ runs in linear time.

To compute $\Delta_{OPM/UD}(s)$, we can compute $\Delta_{OPM/R}(s)$ using the above algorithm on the reverse of s and then $\Delta_{OPM/UD}(s) = \min\{\Delta_{OPM/L}(s), \Delta_{OPM/R}(s)\}$. □

It should be noted that the Lempel–Ziv scheme uses the same decoding algorithm as any other unidirectional OPM scheme, and so the decoding complexity of our method is the same as that of Lempel–Ziv.

5. *Internal Versus External Schemes*

Although a number of bounds have already been given on the relative performance of various pairs of compression methods, we have yet to compare the effectiveness of

the external schemes to the internal schemes. We shall now present a few results of this kind. In order to avoid trivial comparisons, we shall require that both schemes under comparison allow recursion if either does (otherwise the relative performance goes to zero). This will cause us to consider schemes that are not particularly natural (internal schemes with nonoverlapping pointers, etc.). The purpose of comparing, say, the EPM scheme without recursion or overlapping to the CPM scheme without recursion and overlapping is not to propose the CPM scheme without recursion and overlapping as a useful scheme, but rather to give some insight with regards to the relative performance of internal and external schemes. The first theorem of this section considers schemes without restrictions.

THEOREM 13. *For all strings s,*

(*a*) $\frac{2}{3}|\Delta_{CPM}(s)| < |\Delta_{EPM}(s)| \leq |\Delta_{CPM}(s)| + p$,
(*b*) $\frac{1}{2}|\Delta_{OPM}(s)| < |\Delta_{OEPM}(s)| \leq |\Delta_{OPM}(s)| + p$,

regardless of whether topological recursion is assumed. Furthermore, for any real $h > 0$, *there are infinitely many strings over a two-symbol alphabet for which the bound of* $\frac{1}{2} + h$ *can be achieved for* (*b*) *and infinitely many strings over a* $K \geq 2$ *symbol alphabet for which the bound of* $(2K - 1)/(3K - 2) + h$ *can be achieved for* (*a*).[18]

PROOF

(a) The proof of the second inequality is trivial since $\Delta_{CPM}(s)$ may be used as the external dictionary. Let us now demonstrate the bound of $\frac{2}{3}$. In what follows, for strings uv and vw (v may be the null string), $uv - vw$ denotes w and $uv + vw$ denotes v. For a string s, consider $\Delta_{EPM}(s) = s_0 \# s_1$. Write s_0 as $s_0 = \prod_{i=1}^{k} r_i$, where r_1 is the first factor in s_0 and r_i, $2 \leq i \leq k$, is $r_{i-1} - z$, where z is a substring of s_0 satisfying the following two conditions:

(1) z is a factor in s_0 that either overlaps with r_{i-1} or starts directly after r_{i-1} (i.e., z is a factor in s_0 and $r_{i-1} - z$ is well defined).
(2) There is no other factor in s_0 that satisfies condition 1 and extends further to the right in s_0 than z.

Since (by definition) $\Delta_{EPM}(s)$ is a minimal-length compressed form, the partition of s_0 as described above is well defined. Furthermore, by construction the following two facts hold:

(1) The set $\{R_i : r_i$ is a compressed form for $R_i\}$ is a set of nonoverlapping substrings of s.
(2) Each factor in s_0 is, for some i, a substring of the string r_i, r_{i+1}.

It is possible to construct a CPM compressed form t for s from s_1 as follows:

All characters (i.e., nonpointers) in s_1 are left intact. Find a pointer q in s_1 with r_1 as its target, and replace q by r_1. For $2 \leq i \leq k$, find a pointer q in s_1 with a target z satisfying $r_i = r_{i-1} - z$, and replace q by $q' r_i$, where q' is a pointer to $r_{i-1} + z$. All other pointers q in s_1 point to a substring of $r_i r_{i+1}$ for some i and may be replaced by two pointers in the obvious way.

It is possible that for some strings, the substitutions described above cause some pointers to have targets of size p or smaller. If this is the case, we can reduce the size of t by deleting pointers of this kind and substituting in the targets. Similarly, it may be possible to reduce the size of t by finding adjacent pairs of pointers that we created as described above and find a new target such that the pair of pointers can be

[18] Note that this implies that the bound of 2/3 is tight for unbounded size alphabets.

replaced by a single pointer together with less than p characters. Since we are looking for a worst-case ratio (which we show to be tight shortly), we can assume that it is not possible to shorten t in the two ways described above. Having made this assumption, it is not hard to show that for a worst-case ratio it must be that $|s_0| \geq pn$, where n denotes the number of pointers in s_1. Thus, if we let m denote the number of characters in s_1, we have

$$\frac{|\Delta_{\text{EPM}}(s)|}{|\Delta_{\text{CPM}}(s)|} \geq \frac{|\Delta_{\text{EPM}}(s)|}{|t|} \geq \frac{pn + m + |s_0|}{2pn + m + |s_0|}$$
$$\geq \frac{pn + |s_0|}{2pn + |s_0|} \geq \frac{2}{3}.$$

A more careful analysis shows that the above inequality must be a strict inequality (i.e., $>$, not $\geq$).

We now show the bound of $\frac{2}{3}$ to be tight. Let us first see how a bound of $\frac{3}{4}$ may be achieved with a two-symbol alphabet. For n a multiple of p let

$$s_n = \prod_{i=1}^{n/p} a^{ip} b^{n-(i-1)p}.$$

Using the EPM scheme, s_n can be written as

$$a^n b^n \# \prod_{i=1}^{n/p} (n - ip + 1, n + p),$$

and so it follows that $|\Delta_{\text{EPM}}(s_n)| \leq 3n$. On the other hand, if we attempt to factor s_n using the CPM scheme, a shortest compressed form for s_n is

$$a^p b^n \left(\prod_{i=2}^{n/p-1} q_{1,i} q_{2,i} \right) a^n b^p,$$

where $q_{1,i}$ denotes a pointer into a^n and $q_{2,i}$ denotes a pointer into b^n. Hence $|\Delta_{\text{CPM}}(s_n)| \geq 4n + O(p)$, and a bound of $\frac{3}{4}$ follows.

For a $K \geq 2$-symbol alphabet we can generalize the above construction by defining

$$s_n = \prod_{i=1}^{K-1} \left(\prod_{j=1}^{n} a_i^{jp} a_{i+1}^{n-(j-1)p} \right),$$

and the bound of $(2K - 1)/(3K - 2)$ follows. In addition, if we let $K = f(n)$ for any unbounded function f, then the bound of $\frac{2}{3}$ follows for an unbounded alphabet size.

(b) For a string s we can consider $\Delta_{\text{OEPM}}(s) = s_0 \# s_1$ and proceed in a fashion analogous to the proof of part (a), the only difference being that this proof is a bit simpler, since we cannot make any claims about $|s_0|$. This is because with original pointers, pointers indicate the decompressed form of s_0, not s_0 itself; thus s_0 can be very small compared to the number of pointers in s_1. Hence, using the notation of part (a),

$$\frac{|\Delta_{\text{OEPM}}(s)|}{|\Delta_{\text{OPM}}(s)|} > \frac{pn}{2pn} = \frac{1}{2}.$$

This bound may be shown tight (even for two-symbol alphabets), as follows. For $n > 0$ let

$$s_n = \prod_{i=1}^{n/p} a^{ip} b^{n-(i-1)p}.$$

Using an external dictionary of $a^n b^n$, it is easy to see that $|\Delta_{\text{OEPM}}(s_n)| \leq n + O(p \log_2(n))$, regardless of whether topological recursion is used, whereas it is easy to

check that $|\Delta_{\text{OPM}}(s_n)| \geq 2n$, regardless of whether topological recursion is used. Thus the bound of $\frac{1}{2}$ is approached arbitrarily closely as n gets large. □

We now turn our attention to bounds concerning restricted schemes. In particular, we consider overlapping and recursion restrictions.

THEOREM 14. *Let Δ_{INT} denote an internal scheme (CPM or OPM) and Δ_{EXT} an external scheme (EPM or OEPM). Furthermore, if Δ_{INT} and Δ_{EXT} are used together, then both refer to compressed pointer schemes or else both refer to original pointer schemes. Then for all strings s,*

(a) $\frac{2}{3} < |\Delta_{EXT}(s)|/|\Delta_{INT}(s)| \leq \frac{4}{3}$ *when recursion is forbidden.*
(b) $1 \leq |\Delta_{EXT}(s)|/|\Delta_{INT}(s)| \leq \frac{4}{3}$ *when overlapping is forbidden or when both recursion and overlapping are forbidden.*

Furthermore, these bounds are tight.

PROOF

(a) When recursion is forbidden, compressed and original pointers are equivalent, and so without loss of generality we consider the CPM scheme. The fact that $\frac{2}{3}$ is a tight lower bound follows from a proof very similar to that of Theorem 13. We now show that $\frac{4}{3}$ is a tight upper bound. It is not hard to show that we need only consider strings s such that for some string r and integer $k \geq 1$,

$$\Delta_{\text{CPM}}(s) = r \prod_{i=1}^{k} q_i,$$

where q_i denotes a pointer to some substring of r. Using r as the dictionary for the EPM scheme, s can be factored using $k + 1$ pointers. Thus we have

$$\frac{|\Delta_{\text{EXT}}(s)|}{|\Delta_{\text{INT}}(s)|} = \frac{|\Delta_{\text{EPM}}(s)|}{|\Delta_{\text{CPM}}(s)|} \leq \frac{|r| + (k+1)p}{|r| + kp}$$
$$\leq \frac{|r| + 2p}{|r| + p} \leq \frac{4}{3}.$$

The string a^{4p} attains this bound.

(b) It is easy to see that 1 is a tight lower bound. Given $\Delta_{\text{EXT}}(s) = s_0 \# s_1$, a corresponding internal compressed form t for s may be formed by "hoisting up" s_0 into s_1 with the following algorithm (this works independently of whether topological or nontopological recursion is present):

A: Let $t = s_1$ and label all pointers "unmarked."
B: **while** there is an unmarked pointer q in t **do**
Replace q by its target and replace all other unmarked pointers in t that have the same target as q by a marked pointer to their target in t.

The (tight) bound of $\frac{4}{3}$ follows by an argument similar to that given in the proof of part (a). □

6. *Conclusion*

We have investigated various aspects of the macro model for performing data compression by text substitution. Results have included NP-completeness theorems on the complexity of finding the most compact encodings for several different macro schemes, relative performance bounds on many pairs of schemes, and a linear-time algorithm for performing optimum compressions for one of the more practical

schemes. All of the schemes we have presented have efficient linear-time decoding algorithms,[19] and many restricted forms of these schemes have real-time decoding algorithms that require only a small amount of random access memory. The same decoding algorithm may be used independently of whether the compressed form is of minimal length. Thus, NP-completeness results indicated in this paper should not discourage further investigation of these schemes. It seems likely that fast and effective approximation algorithms for compressing strings exist for many of the macro schemes with NP-complete encoding complexities. In addition, a number of further results that we have not discussed in this paper lead to polynomial-time compression algorithms for various restricted forms of these problems.

ACKNOWLEDGMENTS. The authors are grateful to J. D. Ullman for helpful comments and to J. Gallant for his critical reading of the paper and for providing a shorter proof of Theorem 2 for the case $p = 1$.

REFERENCES

1. AHO, A.V., HOPCROFT, J.E., AND ULLMAN, J.D. *The Design and Analysis of Computer Algorithms.* Addison-Wesley, Reading, Mass., 1974.
2. BOYER, R.S. A fast string searching algorithm *Commun. ACM 20*, 10 (Oct. 1977), 762–772.
3. COOK, S.A The complexity of theorem proving procedures. Proc. 3rd Ann. ACM Symp. on Theory of Computing, Shaker Heights, Ohio, 1971, pp. 151–158.
4. GALLANT, J., MAIER, D., AND STORER, J.A. On finding minimal length superstrings. *J. Comput. Syst. Sci. 20* (1980), 50–58
5. GAREY, M.R., JOHNSON, D.S., AND STOCKMEYER, L. Some simplified NP-complete problems. *Theor. Comput. Sci. 1* (1976), 237–267.
6. HAGAMEN, W.D., LINDEN, D.J., LONG, H.S., AND WEBER, J.C. Encoding verbal information as unique numbers. *IBM Syst. J. 11* (1972), 278–315.
7. HAHN, B. A new technique for compression and storage of data. *Commun. ACM 17*, 8 (Aug. 1974), 434–436
8. HUFFMAN, D.A. A method for the construction of minimum-redundancy codes, *Proc. IRE 40* (1952), 1098–1101.
9. KARP, R.M. Reducibility among combinatorial problems. In *Complexity of Computer Computations*, R.E. Miller and J.W. Thatcher, Eds., Plenum Press, New York, 1972, pp. 85–103.
10. KNUTH, D.E. *The Art of Computer Programming, Vol. 1: Fundamental Algorithms*, 2nd ed. Addison-Wesley, Reading, Mass., 1973.
11. KNUTH, D.E., MORRIS, J.H., AND PRATT, V.R. Fast pattern matching in strings. *SIAM J. Comput. 6*, 2 (1977), 323–349.
12. LEMPEL, A., AND ZIV, J. On the complexity of finite sequences. *IEEE Trans Inf. Theory IT 22*, 1 (1976), 75–81.
13. LESK, M.E. Compressed text storage. Unpublished Tech. Memo., Bell Laboratories, Murray Hill, N.J., 1970
14. MCCARTHY, J.P. Automatic file compression. In *International Computing Symposium*, North-Holland, Amsterdam, 1973, pp. 511–516.
15. MCCREIGHT, E.M. A space-economical suffix tree construction algorithm. *J. ACM 23*, 2 (Apr. 1976), 262–272.
16. MAIER, D. The complexity of some problems on subsequences and supersequences. Conf. on Theoretical Computer Science, University of Waterloo, Waterloo, Ont., Can., 1977, pp. 120–129
17. MAIER, D., AND STORER, J.A. A note on the complexity of the superstring problem. Proc. 1978 Conf. on Information Sciences and Systems, Baltimore, Md., 1978, pp. 52–60.
18. MAJSTER, M.E. Efficient on-line construction and correction of position trees. Tech. Rep. TR79-393, Dep. of Computer Science, Cornell Univ., Ithaca, N.Y., 1979.
19. MARRON, B.A., AND DE MAINE, P.A.D. Automatic data compression. *Commun. ACM 10*, 11 (Nov. 1967), 711–715
20. MAYNE, A., AND JAMES, E.B. Information compression by factorising common strings. *Comput. J. 18*, 2 (1975), 157–160

[19] The decoding algorithms presented in Definitions 1–4 are used because they are simple to state, not because they are the most efficient algorithms

21. MORRIS, R., AND THOMPSON, K. Webster's second on the head of a pin. Unpublished Tech. Memo., Bell Laboratories, Murray Hill, N.J., 1974.
22 PRATT, V.R. Improvements and applications for the Weiner repetition finder. Lecture notes, 3rd revision, 1975.
23. RODEH, M , PRATT, V R., AND EVEN, S A linear-time algorithm for finding repetitions and its application to data compression, Tech Rep. No. 72, Dep of Computer Sci., Technicon, Israel, 1976.
24. RODEH, M., PRATT, V.R., AND EVEN, S. Linear algorithm for data compression via string matching. *J. ACM 28,* 1 (Jan 1981), 16–24.
25 RUBIN, F Experiments in text file compression. *Commun. ACM 19,* 11 (Nov. 1976), 617–623.
26 RUTH, S S , AND KREUTZER, P.J. Data compression for large business files. *Datamation 18,* 9 (1972), 62–66
27. SEERY, J B., AND ZIV, J. A universal data compression algorithm· Description and preliminary results. Unpublished Tech. Memo., Bell Laboratories, Murray Hill, N.J., 1977.
28. SEERY, J.B., AND ZIV, J. Further results on universal data compression. Unpublished Tech. Memo., Bell Laboratories, Murray Hill, N.J., 1978.
29. SEIFERAS, J. Subword trees Lecture notes, 1977
30 STORER, J.A NP-completeness results concerning data compression. Tech. Rep. 234, Dep. of Electrical Engineering and Computer Science, Princeton Univ., Princeton, N.J., 1977.
31 STORER, J.A. PLCC—A compiler-compiler for PL1 and PLC users. Tech. Rep. 236, Dep. of Electrical Engineering and Computer Science, Princeton Univ., Princeton, N.J., 1977.
32. STORER, J.A. Data compression: Methods and complexity issues. Ph.D. Dissertation, Dep. of Electrical Engineering and Computer Science, Princeton Univ , Princeton, N.J., 1978.
33. STORER, J.A., AND SZYMANSKI, T.G. The macro model for data compression. Proc. 10th Ann. ACM Symp. on Theory of Computing, San Diego, Calif., 1978 (extended abstract).
34. VISVALINGAM, M. Indexing with coded deltas—A data compaction technique. *Softw. Pract. Exper 6* (1976), 397–403.
35. WAGNER, R A. Common phrases and minimum-space text storage. *Commun. ACM 16,* 3 (Mar. 1973), 148–152
36. WEINER, P Linear pattern matching algorithms. Proc. 14th Annual IEEE Symp. on Switching and Automata Theory, Ames, Iowa, 1973, pp. 1–11.
37. ZIV, J. Coding theorems for individual sequences *IEEE Trans. Inf. Theory IT 24,* 4 (1978) 405–412.
38. ZIV, J , AND LEMPEL, A. A universal algorithm for sequential data compression. *IEEE Trans Inf. Theory IT 23,* 3 (1977), 337–343.
39. ZIV, J , AND LEMPEL, A. Compression of individual sequences via variable-rate coding. *IEEE Trans. Inf. Theory IT 24,* 5 (1978), 530–536.

RECEIVED JULY 1979, REVISED JUNE 1980, ACCEPTED JUNE 1981